POLITECNICO DI TORINO

Corso di Laurea Magistrale

in Ingegneria Gestionale

Tesi di Laurea Magistrale

Modelli Predittivi di Machine Learning applicati all'Analisi Creditizia

Relatore

Prof. Franco Varetto

Candidato

Gianmarco Salvadori

Anno Accademico 2020-2021

INDICE

INTRODUZIONE

Alla base di questo elaborato vi è un'analisi creditizia applicata alle imprese operanti in un particolare settore economico italiano, quello metallurgico. In particolare, l'obiettivo di questa tesi di laurea è quello di realizzare un sistema di Credit Scoring, mediante l'utilizzo di alcuni tra i più importanti e attuali modelli di Machine Learning come la Regressione Logistica, la Rete Neurale e l'XGBoost, al fine di tentare di predire l'evento default/non default societario sulla base dell'affidabilità creditizia delle aziende prese in esame.

La principale motivazione che mi ha spinto ad approfondire tale tema è stata l'interesse nel confrontare quale tra gli algoritmi sopracitati risultasse maggiormente adatto all'obiettivo prefissato in termini sia di prestazioni che di robustezza dei risultati.

La tesi è articolata in sei capitoli: nel primo capitolo viene mostrata una panoramica generale sull'andamento passato e presente del settore metallurgico in Italia, in Europa e nel Mondo. Dopo un breve accenno al tema del Rischio di Credito, i successivi tre capitoli descrivono il funzionamento teorico dei diversi modelli di ML utilizzati nel progetto, mentre il quinto approfondisce alcuni dei loro aspetti peculiari. L'ultimo capitolo, ovvero il cuore della tesi, spiega dettagliatamente, in ogni sua fase, l'intero progetto svolto, dalla scelta del settore economico e creazione del campione di imprese allo sviluppo dei modelli di Machine Learning, confrontandone i pregi, particolarità e le relative problematiche riscontrate. Infine, è stata eseguita un'analisi degli errori commessi con lo scopo di verificare la validità e attendibilità delle performance registrate.

SETTORE METALLURGICO

In Europa e nel Mondo

La metallurgia è la disciplina tecnica relativa allo studio dei metalli, del loro comportamento e dei procedimenti tecnici volti al loro ottenimento e lavorazione. È un importante ramo dell'intera industria europea e include diverse attività, tra le quali: la lavorazione dei minerali, l'estrazione e raffinazione dei metalli dai minerali, la produzione e saldatura di leghe metalliche, la lavorazione dei metalli sotto pressione, la fabbricazione di parti in metallo fuso, la lavorazione termica, termochimica, termomeccanica, il rivestimento superficiale delle parti metalliche con uno strato di altri metalli mediante diffusione di alcune sostanze (metalliche o non metalliche) nello strato superficiale degli oggetti metallici. Attualmente, queste rappresentano circa il 46% e l'11% rispettivamente del Valore totale della Produzione e del PIL dell'Unione Europea.

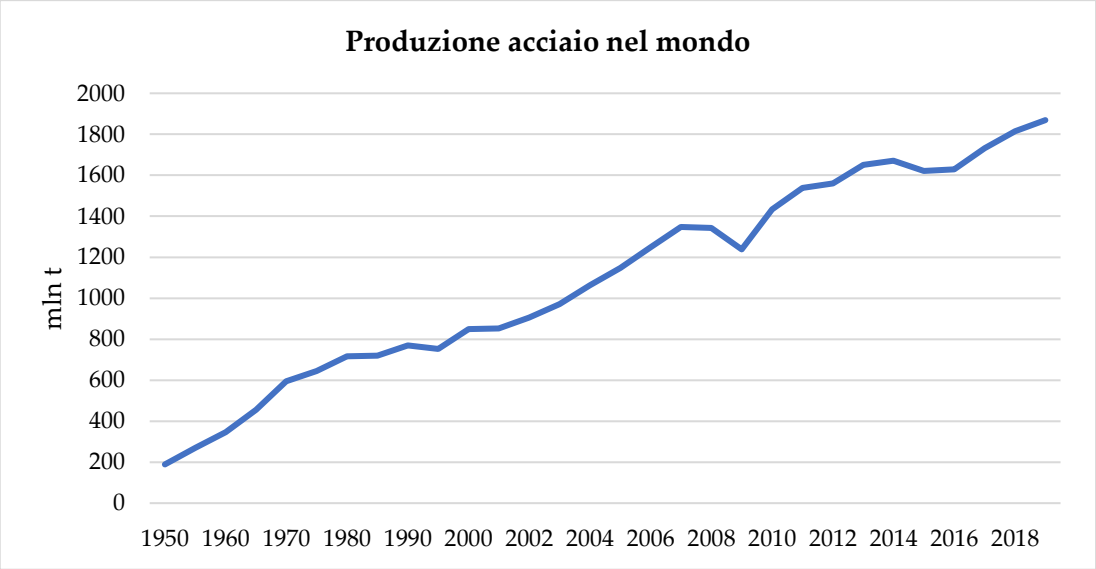
L'industria metallurgica europea è leader mondiale nell'innovazione e sostenibilità ambientale. Essa sostiene direttamente circa 330.000 persone altamente qualificate e, attraverso effetti indiretti e indotti, sostiene fino a 2,2 milioni di posti di lavoro in più. Il settore produce in media circa 170 milioni di tonnellate di acciaio all'anno in oltre 500 siti di produzione all'interno di 23 Stati membri dell'UE. L'acciaio, essendo uno dei materiali più versatili al mondo e riciclabile al 100%, risulta essere la colonna portante per lo sviluppo di tecnologie innovative, ad esempio riguardanti la riduzione della CO2, nonché una parte fondamentale per un'economia circolare e sostenibile di successo.

Nel corso del 2019, l'occupazione in Europa è stata stabile nonostante una domanda sul il mercato interno significativamente in calo. L'espansione, in corso ormai da diversi anni, dei settori che utilizzano l'acciaio è giunta ad un arresto, registrando una forte riduzione delle performance del mercato di tale materiale; in particolare, il consumo è sceso del 6,23% a 192,6 milioni di tonnellate e le importazioni sono drasticamente calate di circa il 7,25%, tornando ai livelli del 2015-2016. Mentre le precedenti stime di crescita suggerivano che ci sarebbe stata una probabile inversione di rotta verso un ritorno all'espansione nel 2020, attualmente, con la pandemia di coronavirus, questo sembra altamente improbabile.

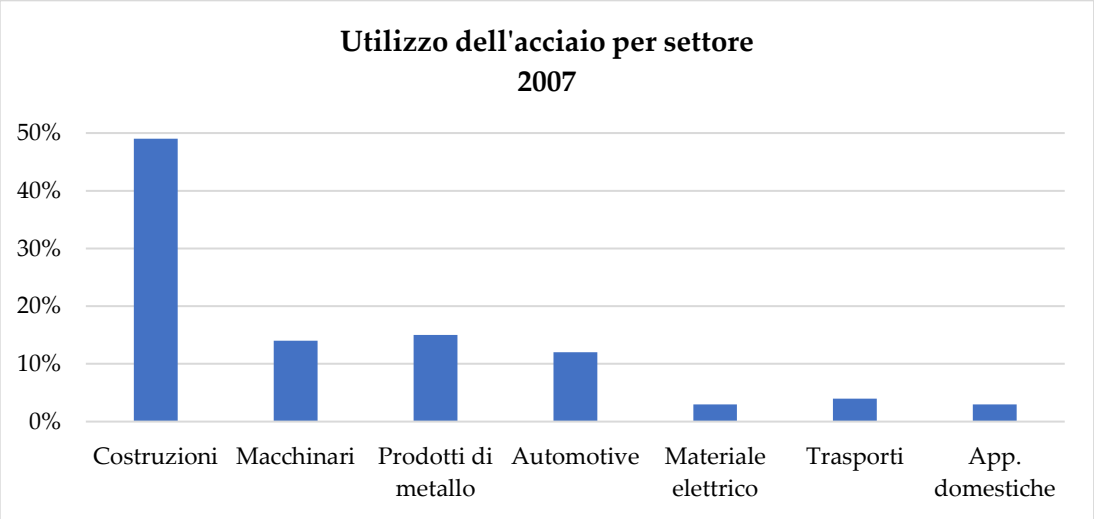
Produzione

Attualmente, l'acciaio è la spina dorsale dell'evoluzione e del progresso della società; le città "intelligenti" di domani saranno costruite prevalentemente con questo materiale in quanto risorsa infinitamente riciclabile e riutilizzabile, volta a ridurre il pesante carico sulle risorse naturali della Terra.

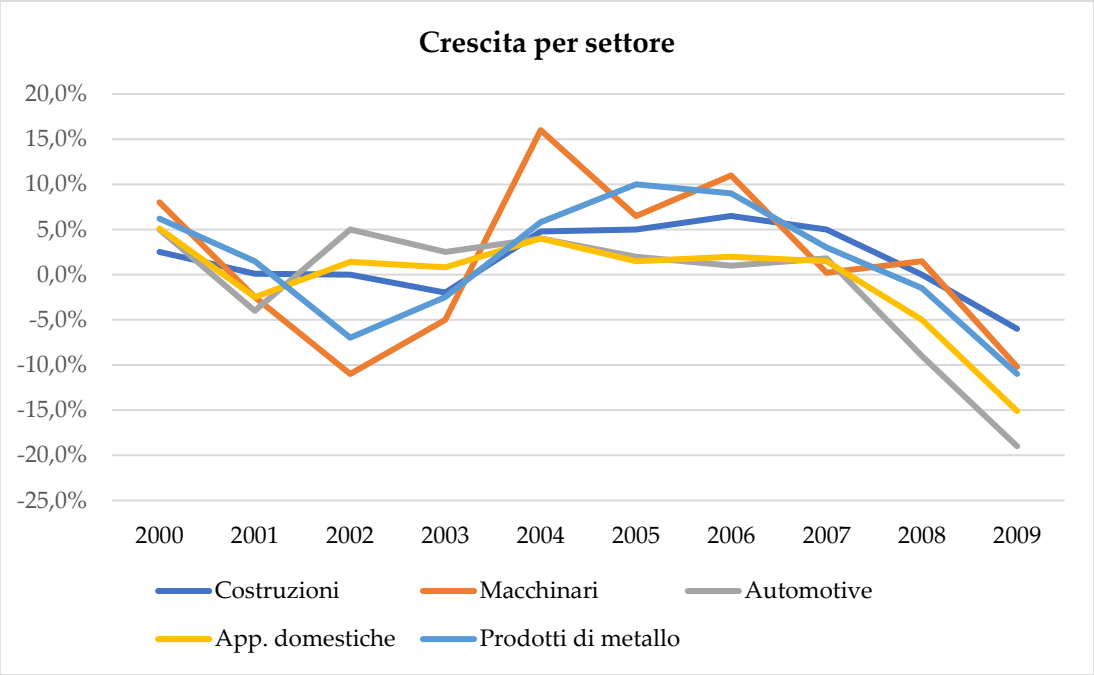
Dai seguenti dati si evince la notevole importanza che ha acquisito nel tempo:



Dall'andamento della curva si possono individuare tre distinti intervalli; dal secondo dopoguerra all'inizio degli anni 70' si registra la prima e significativa fase di crescita del settore, con un tasso annuo di crescita composto (CAGR) del 5,8%; il secondo, che arriva fino ai primi anni 2000, al contrario fu un periodo di relativa stagnazione contraddistinta da una crescita pressoché nulla (0,7%), mentre, nei successivi 7 anni, avvenne il boom definitivo del settore che performò di oltre l'8,4%. Il picco negativo maggiormente rilevante ci fu tra il 2008 e il 2009 a causa della grande crisi economica verificatasi negli Stati Uniti d'America.

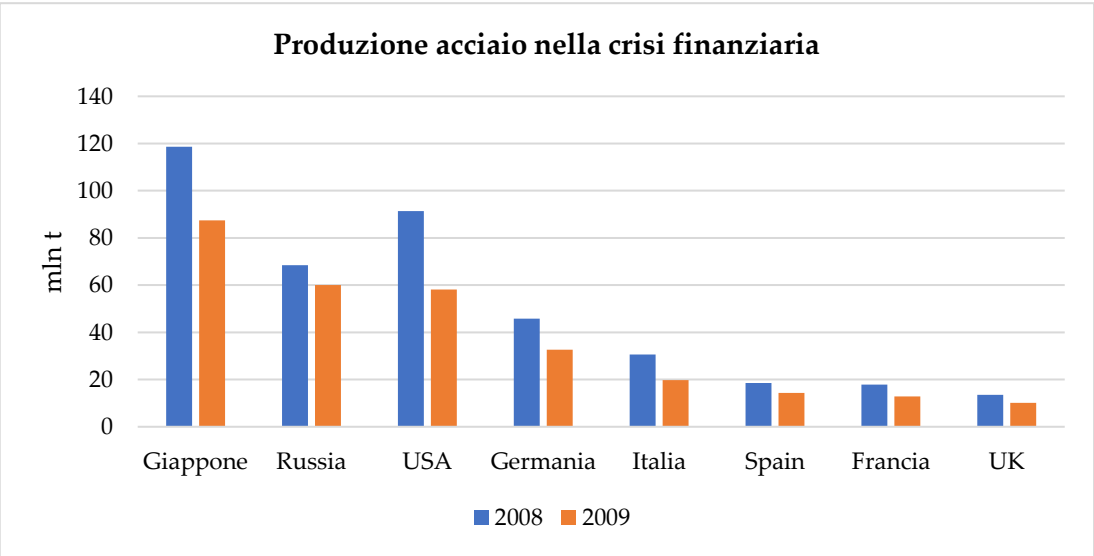


Nata come crisi bancaria, essa si espanse anche a molti altri settori come quello delle costruzioni, automotive e dei macchinari meccanici -tre dei settori che maggiormente utilizzano l'acciaio- andando, in questo modo, a colpire duramente l'intero settore metallurgico in termini di contrazione della domanda del materiale, tagli alla produzione nella maggior parte dei paesi e riduzione dell'occupazione e dei prezzi.



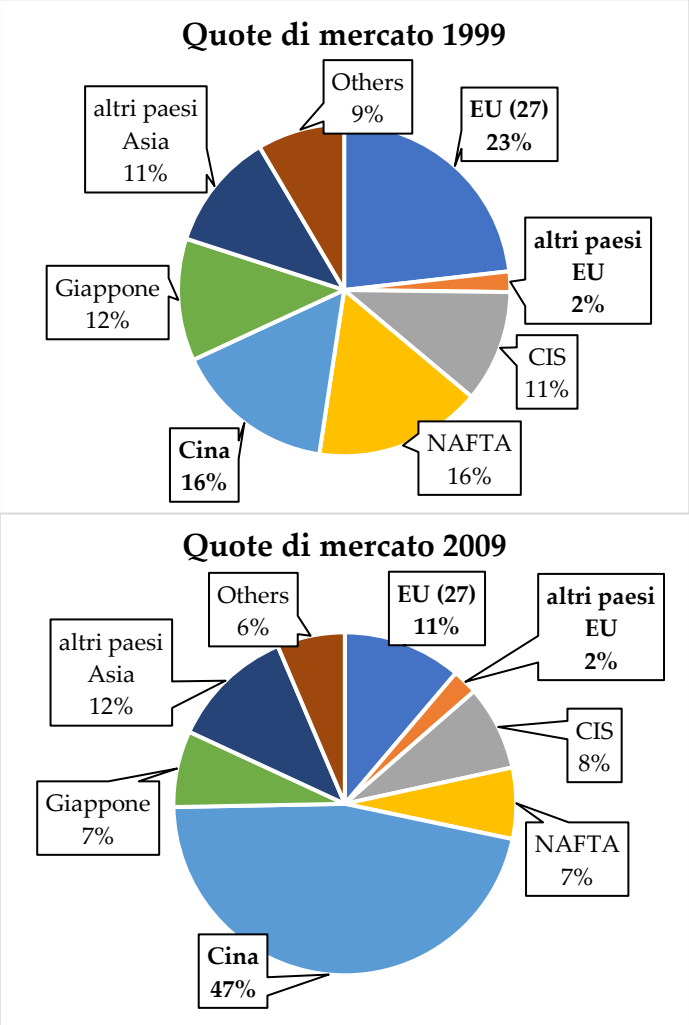
Come si nota dal grafico, molti dei principali consumatori di acciaio grezzo stavano attraversando una fase di decrescita anche durante gli anni precedenti lo shock finanziario ma questo ha inevitabilmente ridotto ulteriormente la disponibilità di capitali a disposizione delle imprese, causandone una forte contrazione degli investimenti nei settori sopra citati.

Inizialmente, i paesi più danneggiati furono gli USA, Giappone e molti paesi europei:



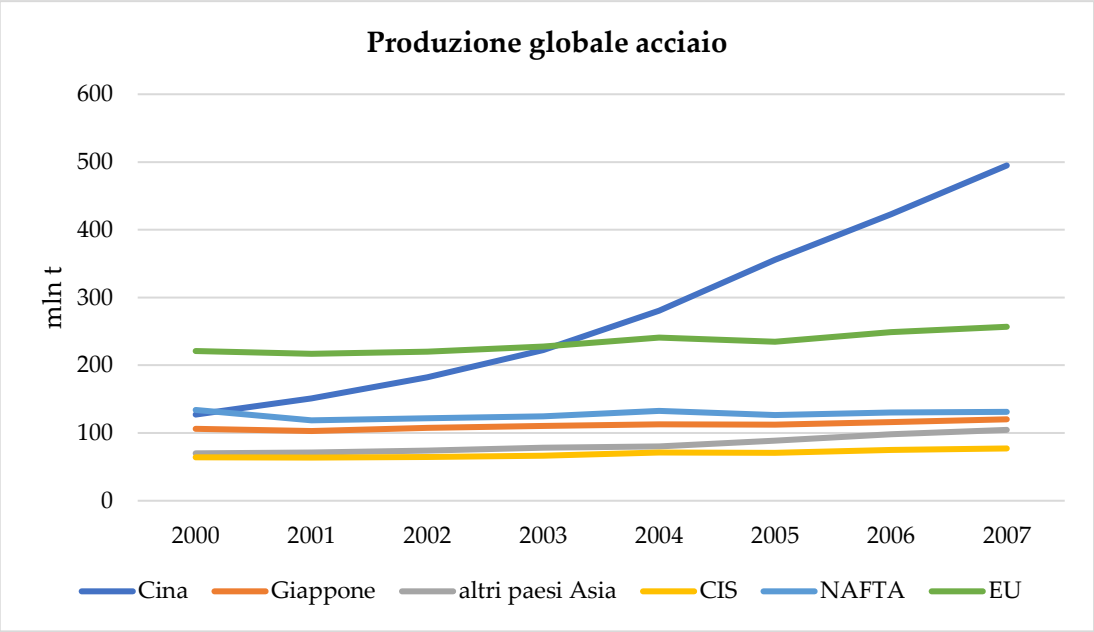
In particolare, l'Italia ridusse la propria produzione del 35,3% (da oltre 30 milioni di tonnellate a poco meno di 20 milioni); la Germania, da sempre maggior produttrice di acciaio grezzo in Europa, nel 2009 ne realizzò 32,7 milioni di tonnellate a fronte dei quasi 46 milioni dell'anno prima, mentre il calo maggiore lo registrarono gli USA con una produzione ridotta di oltre il 36%.

Storicamente, l'Europa è sempre stata in prima linea per quanto riguarda la produzione nel settore metallurgico; tuttavia, nei primi anni 2000 la Cina ha iniziato a prendere enormi quote di mercato e oggi, per competerle, tutti gli altri paesi devono necessariamente aumentare gli sforzi per sviluppare innovazioni sui prodotti, processi, infrastrutture e sulle capacità produttive.



I grafici mostrano, in termini di quote di mercato di acciaio prodotto, la differenza che vi era a fine anni 90' con quella immediatamente successiva alla grande crisi finanziaria dei subprime. Si nota come nella prima ci fosse una distribuzione equa delle quote, nella quale l'Europa risultava essere la regione con maggiore presenza nel

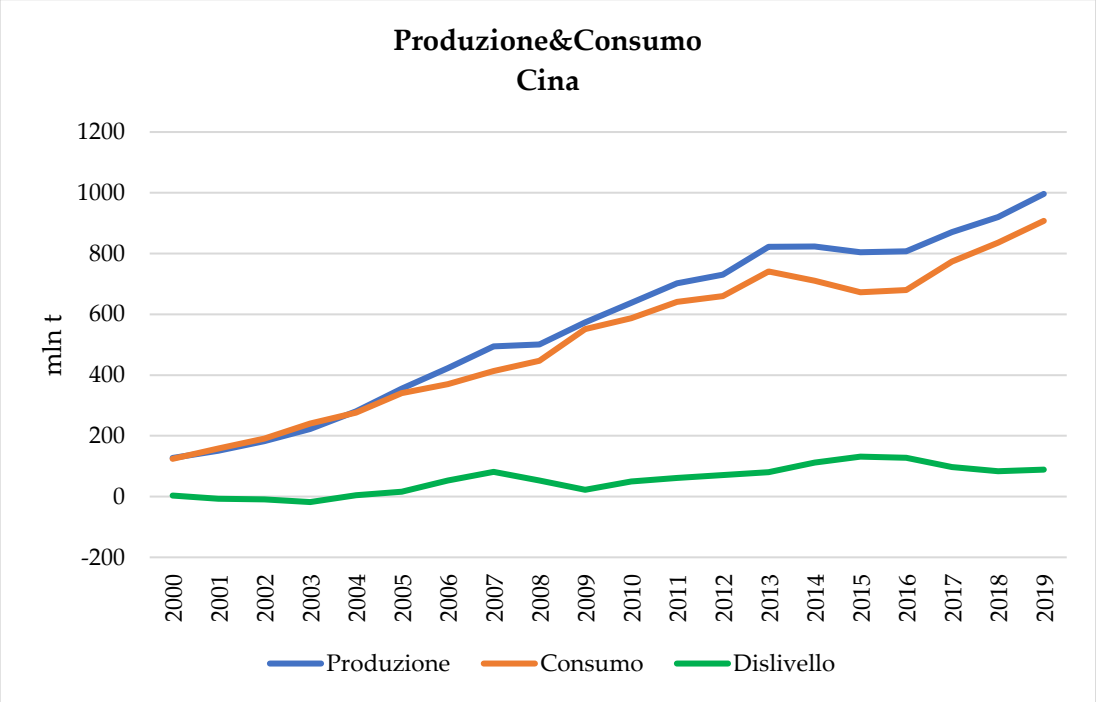
settore con oltre 220 milioni di tonnellate di acciaio grezzo prodotto.



Negli anni immediatamente successivi lo scenario mondiale cambiò radicalmente; con una produzione globale in costante aumento, la Cina prese definitivamente e stabilmente il controllo del settore grazie alla crescita esplosiva della sua economia interna, a ingenti investimenti incentrati sullo sviluppo delle industrie pesanti (produzione macchinari e attrezzature di trasporto) e anche a un livello di urbanizzazione senza precedenti, al quale seguì un aumento vertiginoso della domanda di abitazioni e infrastrutture come strade, ferrovie, porti e dunque di acciaio. Il tutto portò ad una produzione globale, a fine 2007, di 1,35 miliardi di tonnellate, di cui il 36% di manifattura cinese.

Consumi

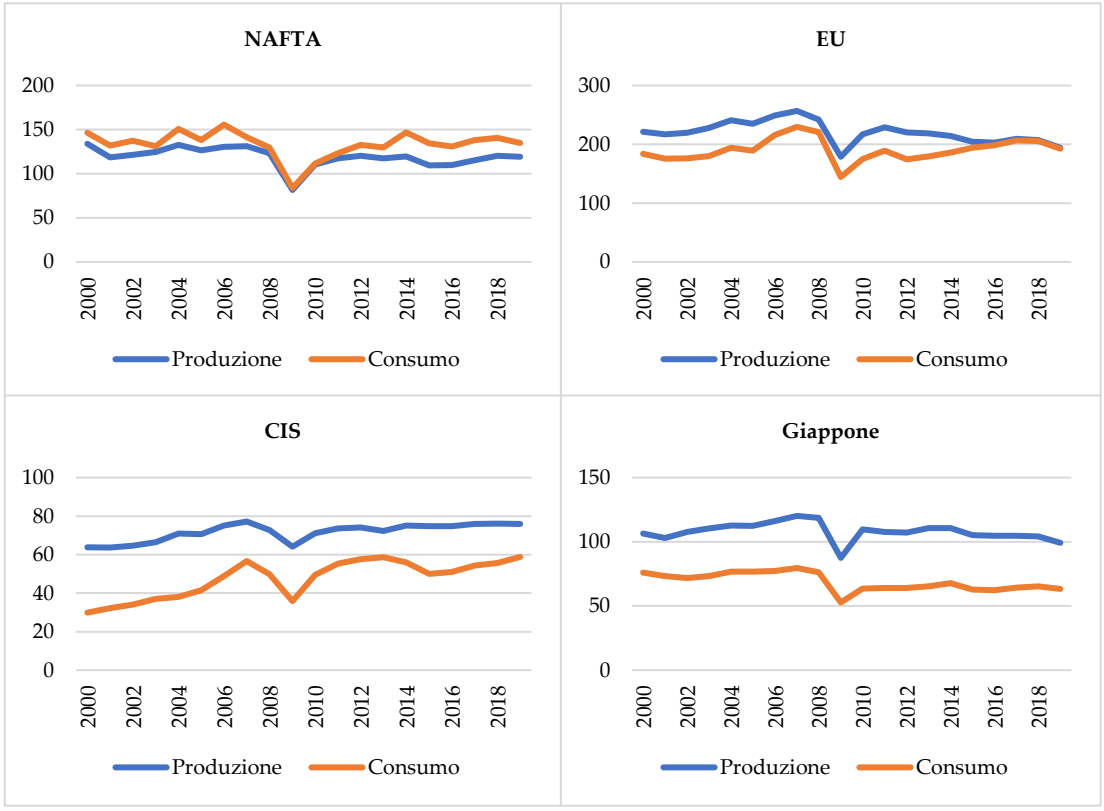
La situazione rimase tale fino alla crisi finanziaria; nel 2008, il tasso di crescita annuale della Cina rallentò provocando un progressivo calo dei consumi di acciaio grezzo e, conseguentemente, l'industria siderurgica rimase intrappolata in un eccesso di capacità**.** Gli effetti della crisi non si manifestarono immediatamente in termini di calo della produzione/offerta (come avvenne per i paesi EU, USA e Giappone) ma, al contrario, in termini di domanda di acciaio; infatti, tra il 2008 e il 2012 la Cina continuò ad aumentare la produzione (573,6 mln t nel 2009, +14,65% rispetto all'anno precedente) nonostante i consumi interni e globali iniziassero ad andare a rilento.

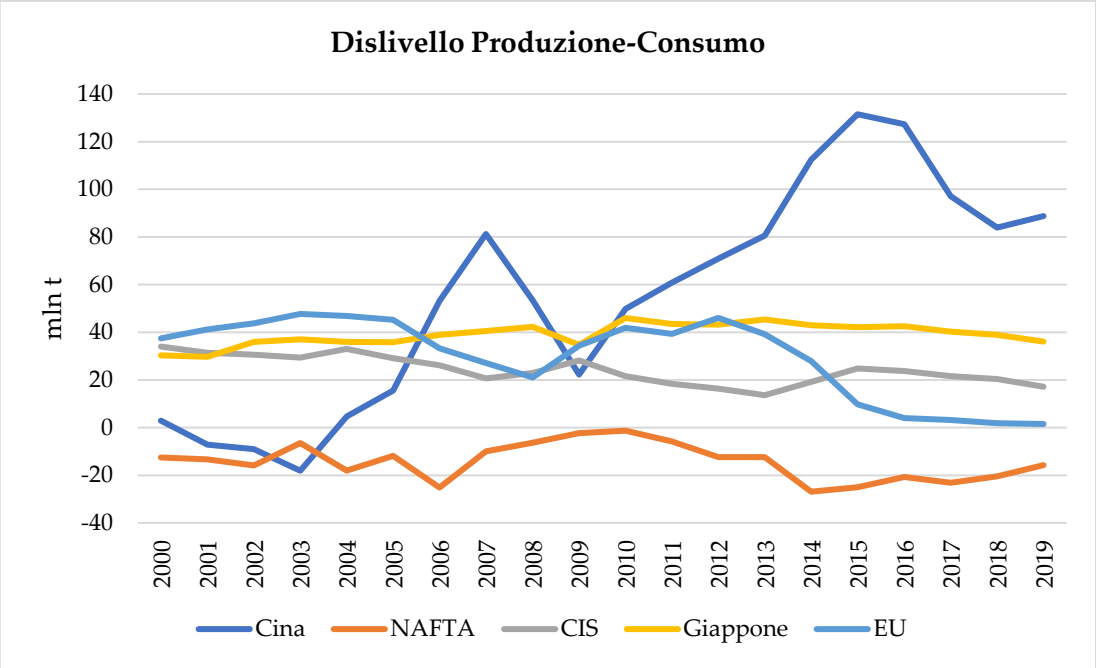


Un primo allarme del problema di sovracapacità avvenne esattamente nell'anno dello shock, ma la situazione degenerò definitivamente nel 2013 quando la Cina raggiunse il più alto disallineamento (+80,6 mln t) tra produzione e consumo interno, vincolato ad una produzione in aumento (822

mln t). Nei due anni successivi il paese non riuscì a adeguare significativamente le capacità produttive, rimaste invariate, e il quadro, dunque, divenne ancora peggiore arrivando a un divario di +131,5 mln t nel 2015; solamente alla fine dell'anno la produzione scese e raggiunse 803,8 mln t, registrando un calo del 2,31% rispetto all'anno precedente. Fu il primo calo della produzione dal 1982.

I problemi riscontrati dalla Cina si estesero facilmente all'intero mercato metallurgico, essendo essa titolare di quasi il 50% della produzione globale di acciaio:





Nell'ultimo decennio, gli andamenti di produzione e consumo di acciaio sono molto simili per due delle quattro aree prese in esame: CIS (Russia, Kazakistan, Armenia, Azerbaigian, Bielorussia, Moldavia) e Giappone; nonostante la scala giapponese sia di gran lunga superiore a quella dei paesi del CIS, entrambe sono caratterizzate da una produzione predominante sul consumo del prodotto, essendo queste estremamente focalizzate sul lato export.

L'Europa tiene anch'essa un'offerta superiore ai consumi ma, negli ultimi 5 anni, le politiche di controllo interno dei paesi membri hanno permesso alla regione di tenere al minimo il divario tra le due grandezze (solamente +1,9 mln t di acciaio rimanente nel 2018, corrispondente allo 0,91% della produzione totale EU).

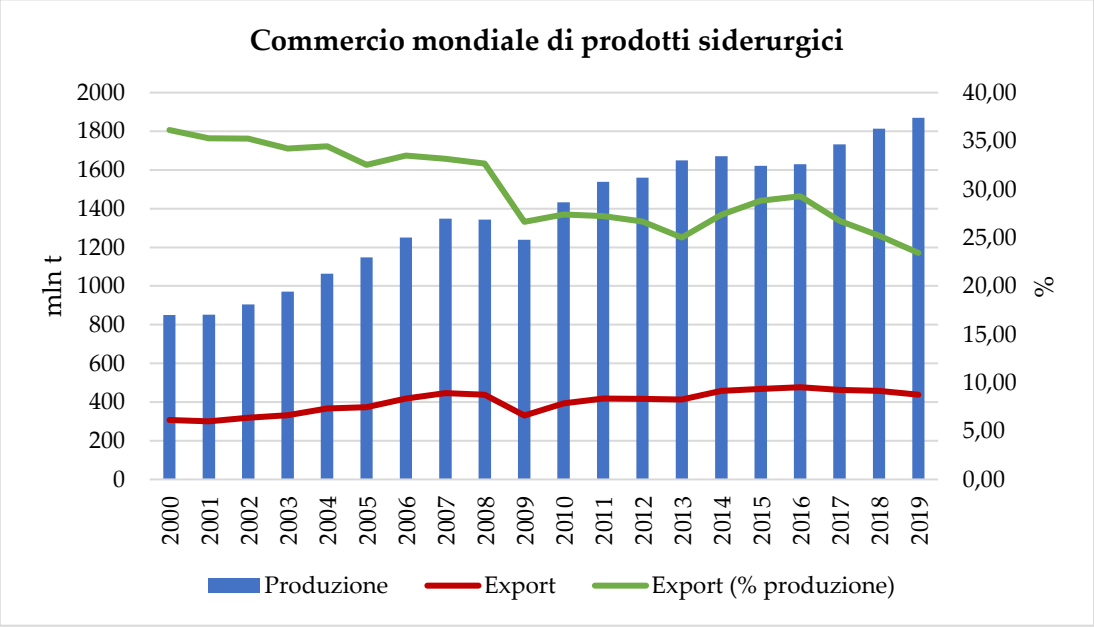
USA, Canada e Messico (NAFTA) mantengono costantemente un livello di consumo leggermente maggiore rispetto alla produzione interna, la quale negli ultimi due anni ha subito un rallentamento a causa della riduzione dei prezzi dell'acciaio a seguito dell'annuncio dell'introduzione dei dazi da parte

dell'amministrazione Trump. Al contrario delle precedenti, questa regione è necessariamente molto presente sul mercato internazionale delle importazioni.

Infine, è importante sottolineare come, a differenza della Cina, queste ultime aree siano riuscite a reagire prontamente allo shock finanziario del 2007, affrontando il drastico calo della domanda di acciaio con uno altrettanto deciso delle quantità prodotte.

Import-Export

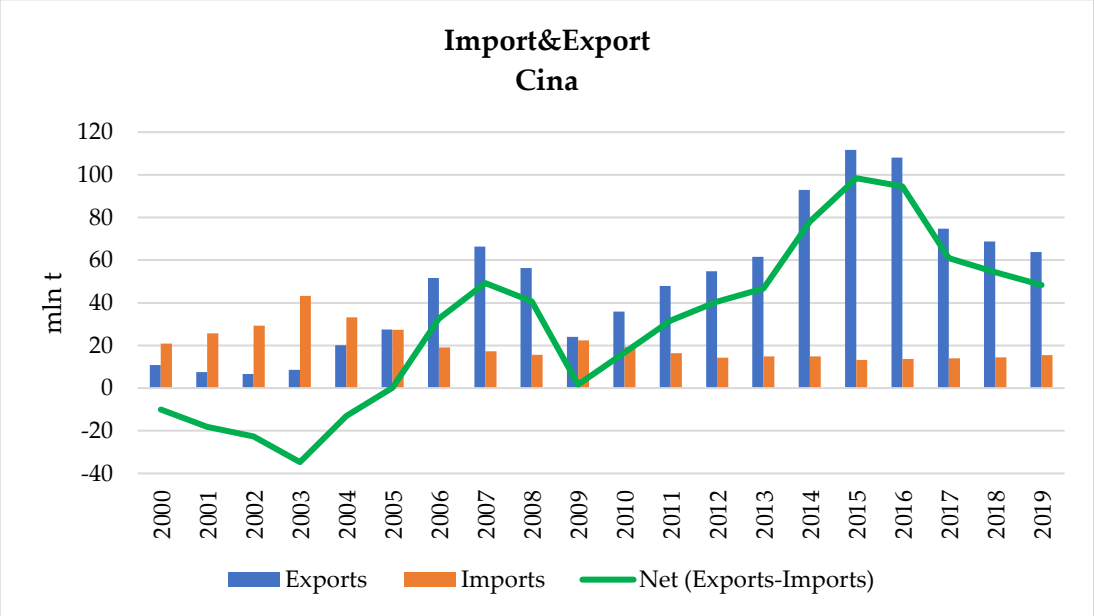
Oltre ai livelli di produzione-consumo, è importante analizzare anche il commercio mondiale di acciaio grezzo per ogni area al fine di comprendere le strategie e misure adottate da ciascuna di esse.



Come si vede dal grafico degli ultimi 20 anni, la produzione globale ha sempre registrato un tasso di crescita significativamente superiore a quello dell'export; inoltre, la curva rappresentante la relazione tra esportazioni e offerta risulta

essere in una chiara fase decrescente a partire dal 2016, evidenziando un trend negativo della prima grandezza.

Tra tutte le regioni, la Cina è certamente quella che maggiormente pilota la curva dell'export. La rapida espansione della sua economia interna le permise di aumentare esponenzialmente la produzione di prodotti siderurgici tanto che essa passò dall'essere il paese con il più alto livello di importazioni di acciaio nel 2003 (43,2 mln t di imports, 8,5 mln t di exports) a diventare, nel 2005, un esportatore netto di tali prodotti; questo quadro si consolidò definitivamente a partire dal 2006 quando la Cina sostituì il Giappone sia come il più grande paese esportatore di acciaio sia come il più grande paese esportatore netto di acciaio al mondo.

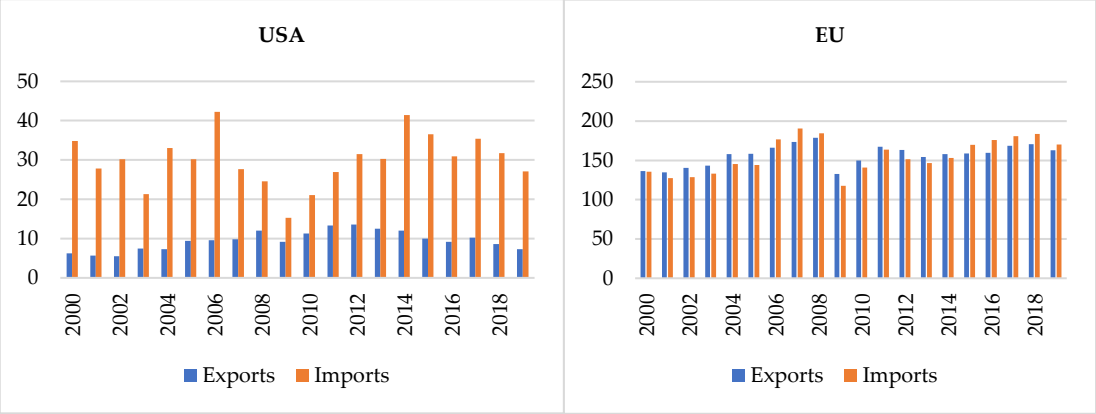


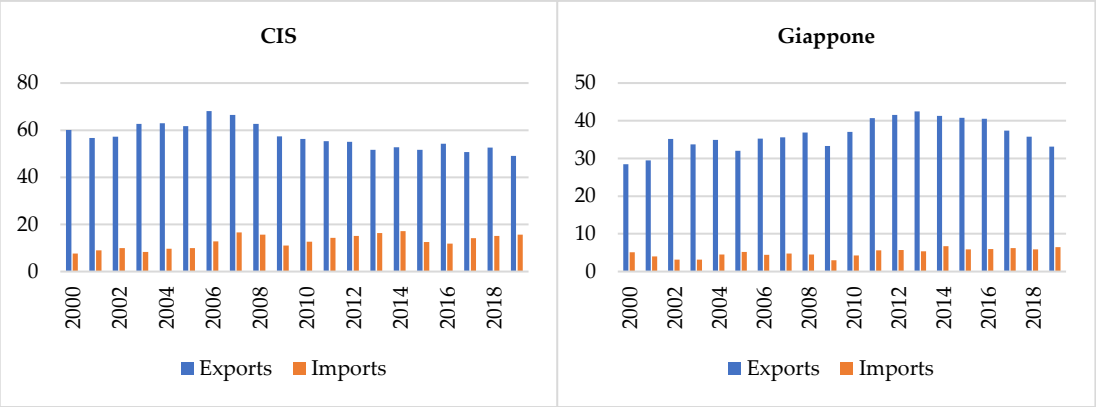
Il rallentamento mondiale dovuto alla crisi finanziaria fece crollare le esportazioni di acciaio in moltissimi paesi e segnò un punto di flessione per l'economia cinese nel momento in cui il cambiamento strutturale -da importatore a esportatore- stava producendo benefici, ad esempio in termini di salari (i quali erano in aumento); le esportazioni scesero velocemente fino a

raggiungere, nel 2009, lo stesso livello delle importazioni (poco più di 22 mln t). Con una produzione in costante aumento e una domanda a ribasso, il crollo delle esportazioni acutizzò molto il problema della sovracapacità, come accennato in precedenza.

In risposta alla crisi, il governo cinese decise di implementare un pacchetto di stimolo fiscale su larga scala e delle politiche espansive al fine di facilitare il mercato export. Sebbene da un lato tali misure contribuirono ad alleviare le pressioni di una produzione in eccesso, dall'altro, avendo aumentato la dipendenza dalle vendite globali, inasprirono molto i rapporti commerciali con gli altri paesi produttori di acciaio. La Cina raggiunse un picco di esportazioni pari a 111 mln t di acciaio nel 2015, corrispondenti a quasi il 150% dell'intera quantità prodotta americana in quell'anno.

Negli ultimi anni il paese ha dovuto necessariamente avviare una ristrutturazione dell'intera industria siderurgica incentrata sulla riduzione della quantità prodotta (chiedendo una riduzione di 150 mln t nei successivi 5 anni) e sull'aumento degli standard tecnici del settore stesso che dovranno portare a un miglioramento dei processi produttivi in termini di consumo di energia, acqua e altre risorse per unità prodotta.





Dalla curva produzione-consumo relativa alla NAFTA vista precedentemente si evince facilmente che gli Stati Uniti d'America, i quali guidano l'andamento di tale curva, sono un paese con una forte attitudine alle importazioni avendo un Net costantemente negativo sull'arco di tempo considerato. Mentre il livello dell'export è generalmente stabile con una media di circa 9,5 mln t l'anno, quello dell'import risulta essere molto variabile e con una sensibilità all'ambiente esterno decisamente maggiore; infatti, questo passa velocemente da un picco di 42,2 mln t nel 2006 a un crollo di oltre il 63% durante lo shock finanziario e ancora a una crescita importante negli anni successivi.

Al contrario, il Giappone e i paesi del CIS sono aree ad alta esportazione avendo una produzione interna di gran lunga superiore ai consumi (vedere i grafici nella sezione "consumi" precedente). Tuttavia, entrambe sono immerse in un trend decrescente per quanto concerne proprio le esportazioni; dal 2013, il Giappone ha perso oltre il 22% di quote export e, ad oggi, esporta 33,1 mln t di acciaio (pari al livello raggiunto durante la crisi finanziaria). Il CIS è, invece, in continua caduta dal 2006 con una perdita sul lato export di circa il 28%.

A causa della contrazione delle economie di paesi membri, l'Europa ha tenuto una produzione di acciaio grezzo a ribasso durante gli ultimi 10 anni in modo tale che ricalchi l'andamento dei nuovi consumi. Tale situazione è ripetuta

anche per i livelli di import-export, i quali nel post shock finanziario vengono mantenuti vicini con un dislivello medio di poco superiore ai 9 mln t all'anno a favore delle importazioni.

In Italia

Uno dei fattori maggiormente rilevanti per lo sviluppo economico italiano nel secondo dopoguerra è stato la rapida crescita dell'industria manifatturiera e, in particolare, dell'export manifatturiero. A contribuire a tale crescita furono il Piano Marshall (1948-51) -un programma sponsorizzato dagli Stati Uniti per rigenerare le economie del dopoguerra dell'Europa occidentale-, la fondazione della Comunità Europea del Carbone e dell'Acciaio (CECA) del 1952 e la nascita della CEE (Comunità Economica Europea) il primo gennaio 1958. Quest'ultima contribuì fortemente alla liberalizzazione del commercio e l'abbondanza di manodopera alimentò la crescita delle imprese industriali su tutto il territorio.

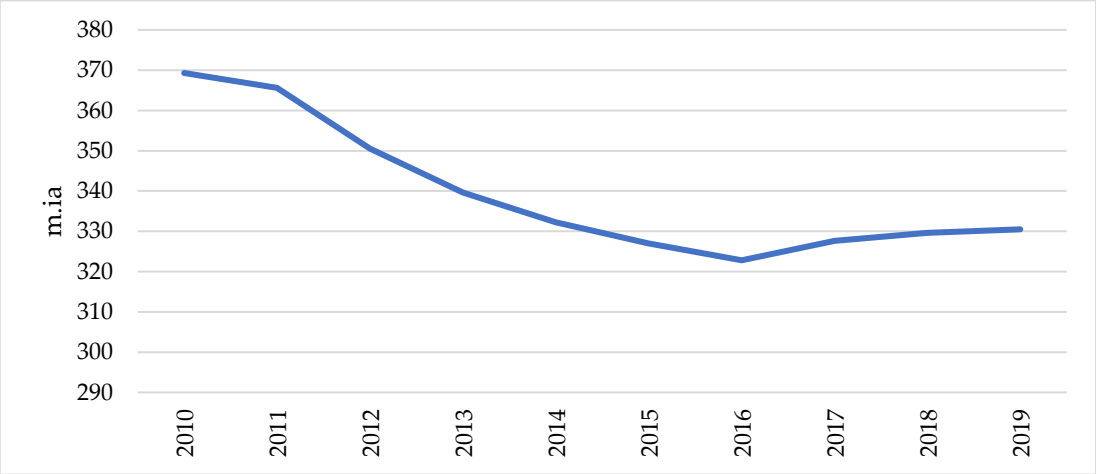
Il materiale che più di tutti ha trasformato l'economia italiana è stato l'acciaio; si sviluppò così rapidamente tanto che, agli inizia degli anni 80', rappresentava oltre il 21% della produzione della CEE e costituiva la spina dorsale dell'industria metallurgica e metalmeccanica. Queste conobbero la loro massima fioritura tra il 1951 e il 1975, quando le esportazioni meccaniche ebbero un boom -aumentando di 20 volte- e la forza lavoro impiegata nelle industrie raddoppiò. L'industria siderurgica, dato il calo degli ultimi decenni del secolo scorso, fu privatizzata tra il 1992 e il 1997.

Attualmente, il settore metallurgico italiano è composto da circa 5000 imprese concentrate principalmente nel nord del paese, soprattutto nella Pianura

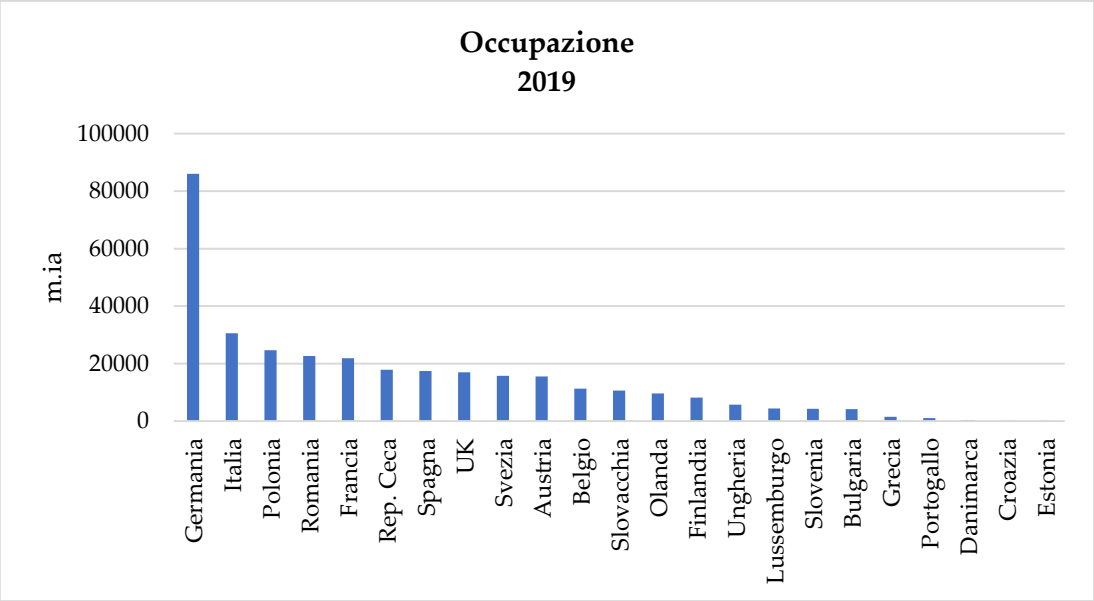
Padana; il centro (terni) e il sud Italia (Taranto) ne ospitano alcune che sono riuscite a imporsi anche nella compagine internazionale.

Nel 2019, l'Italia si è posizionata 11-esima nel mondo per produzione di acciaio con oltre 23 milioni di tonnellate che rappresentano circa l'1,24% di quella globale; a livello europeo detiene quasi il 15% delle quote di mercato, dietro solamente al 25,1% della Germania. Sul lato della domanda, essa registra una crescita nel consumo di acciaio grezzo (25,5 mln t nel 2019) in netta controtendenza rispetto all'andamento generale europeo. Al contrario, come la maggior parte dei paesi europei, negli ultimi anni ha tentato di limitare il divario tra esportazione e importazioni, le quali ammontano rispettivamente a 17,9 e 20,1 mln t nell'ultimo anno preso in considerazione.

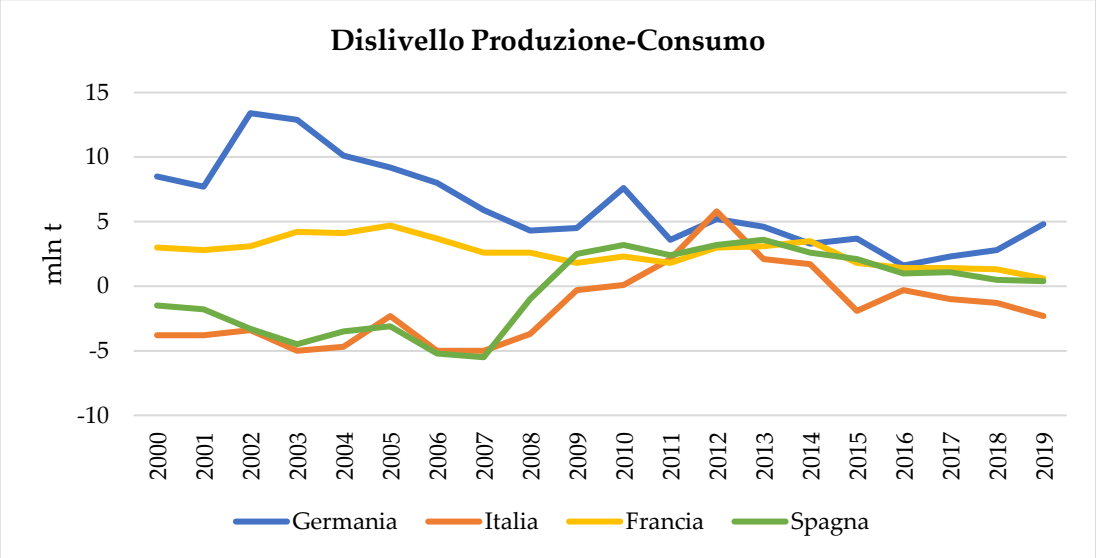
I dati sottostanti mostrano l'evoluzione dell'occupazione in Europa, tenendo in considerazione solamente il conteggio dei dipendenti diretti.

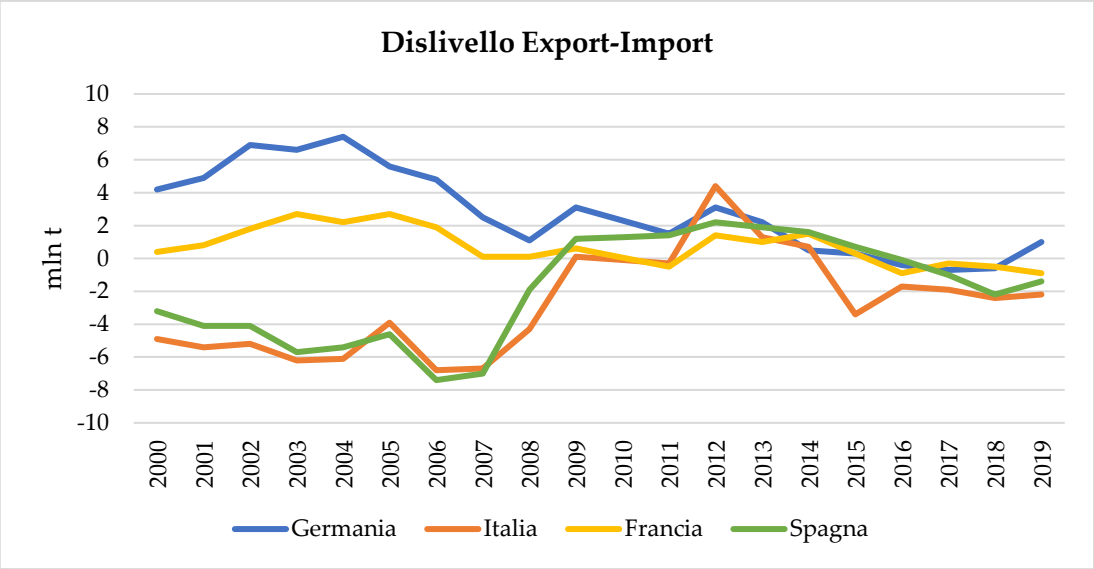


L'occupazione in Europa è rimasta sostanzialmente invariata negli ultimi due anni ma, se confrontata con quella del decennio, si nota un trend fortemente decrescente nel numero di dipendenti che operano direttamente nel settore metallurgico tra il 2011 e il 2016, registrando un -11,7% in tale periodo.



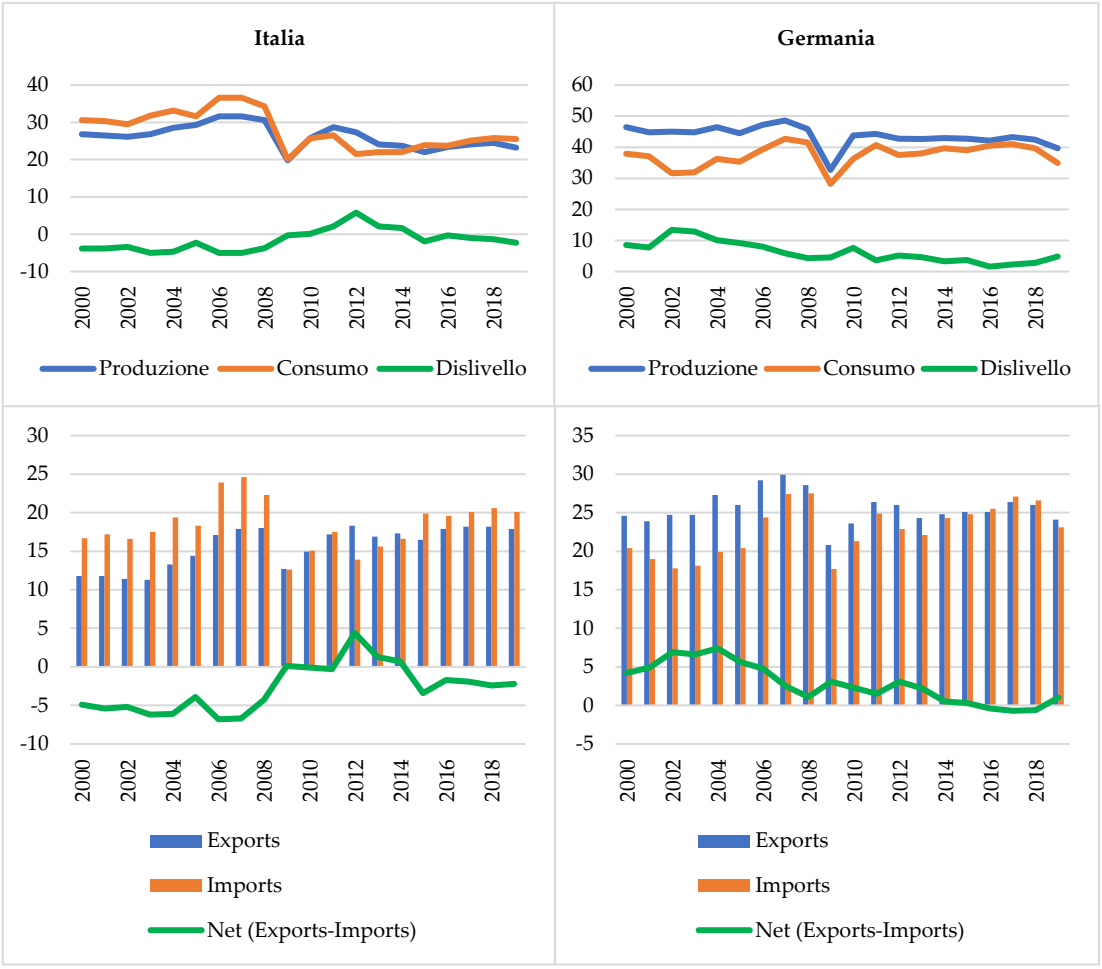
Con i suoi 30 mila lavoratori, l'Italia si posiziona seconda come nazione per occupazione diretta, dietro solamente alla Germania con circa il triplo dei dipendenti.





Come visto precedentemente, negli ultimi anni l'Europa, nel suo complesso, ha dovuto rivedere la sua posizione all'interno del settore siderurgico e sta comprimendo la sua offerta al fine di adeguarsi ai nuovi livelli di consumo. I due grafici mostrano l'evolversi di questo cambiamento; dal periodo post-crisi, la forbice tra produzione e consumo si è man mano assottigliata e addirittura per alcune nazioni, come Francia e Spagna, si è azzerata. Si riesce a vedere con ancor più evidenza nel secondo grafico nel quale, dal 2017, la differenza tra esportazioni e importazioni è costantemente rientrata tra i +/- 2 mln t.

Con i prossimi grafici si rappresenta nel dettaglio il confronto tra Italia e la Germania sotto il punto di vista "produzione-consumo" e "export-import":



I due paesi hanno andamenti simili nella forma delle curve ma significativamente diversi nella sostanza. Agli inizi degli anni 2000, questi hanno intrapreso strategie diametralmente opposte; nonostante entrambe operino molto sia sul mercato delle esportazione sia in quello delle importazioni, l'Italia decise di ricoprire un ruolo da importatore netto, avendo una produzione interna costantemente inferiore al consumo di acciaio. La Germania, al contrario, si impose come primo paese esportatore europeo, riuscendo a mantenere tale status ancora oggi. A fronte della crisi finanziaria, Italia e Germania hanno reagito prontamente, così come la maggior parte degli altri paesi europei, tagliando la produzione ai primi segnali di una domanda in calo.

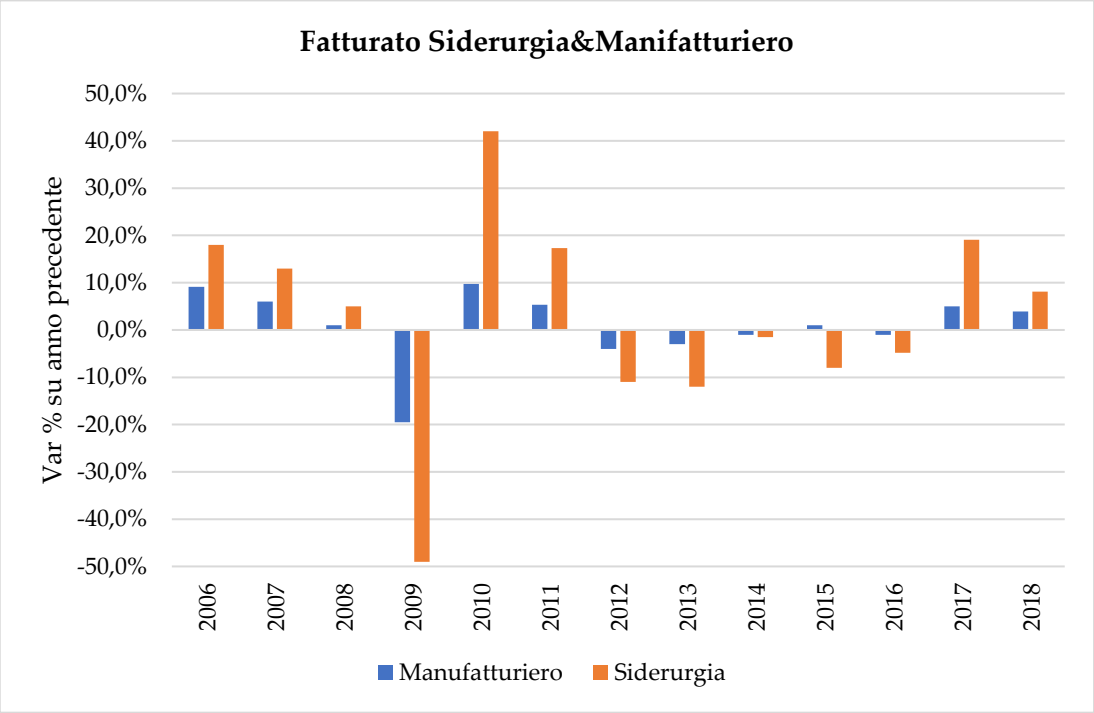
Infine, è importante notare come la curva Net (Ex-Imp) ricalchi quella del dislivello tra produzione e consumo; infatti, a consumi superiori alla quantità prodotta tendenzialmente è associato un maggiore import rispetto all'export, e viceversa. Quest'ultimo caso lo si può vedere chiaramente sul grafico italiano, nel quale tra gli anni 2012/2014 -unici con una produzione superiore ai consumi- si ha una presenza italiana maggiore nel mercato dell'export rispetto all'import.

Fatturato e Valore Aggiunto

Il settore metallurgico è sempre stato molto importante all'interno dell'economia italiana producendo svariati input per altre aree dell'industria (come quella automobilistica, costruzioni, edilizia) e incidendo fortemente sia nel PIL sia nell'occupazione tanto che la produzione annuale di acciaio venne utilizzata per anni come indicatore per misurare lo sviluppo economico del paese.

Tradizionalmente, i problemi ricorrenti che gravano sul settore sono la concorrenza sempre più forte dei paesi emergenti (soprattutto quelli del sud est asiatico) e quelli di natura ambientale rappresentati dall'impatto dell'inquinamento nelle zone sulle quali sono stati installati gli impianti di produzione dell'acciaio. Inoltre, ad oggi l'Italia opera in un contesto politico internazionale molto incerto su diversi fronti: i dazi statunitensi sulle importazioni di prodotti siderurgici imposti all'EU, Brexit, le forti oscillazioni dei prezzi delle materie prime e le sempre più crescenti tensioni commerciali con Cina e gli stessi USA. Ad aggiungersi a tutto ciò c'è una crescita del PIL dello 0,8% nel 2018 e dello 0,3% nel 2019, investimenti ed esportazioni a rilento e una bassa inflazione.

Approfondendo il contesto nazionale, il fatturato prodotto dal settore siderurgico contribuisce a quello dell'intera industria manifatturiera direttamente per il 4% e indirettamente per una quota vicina al 35% attraverso le varie attività dei settori che utilizzano il prodotto output della siderurgia[1].



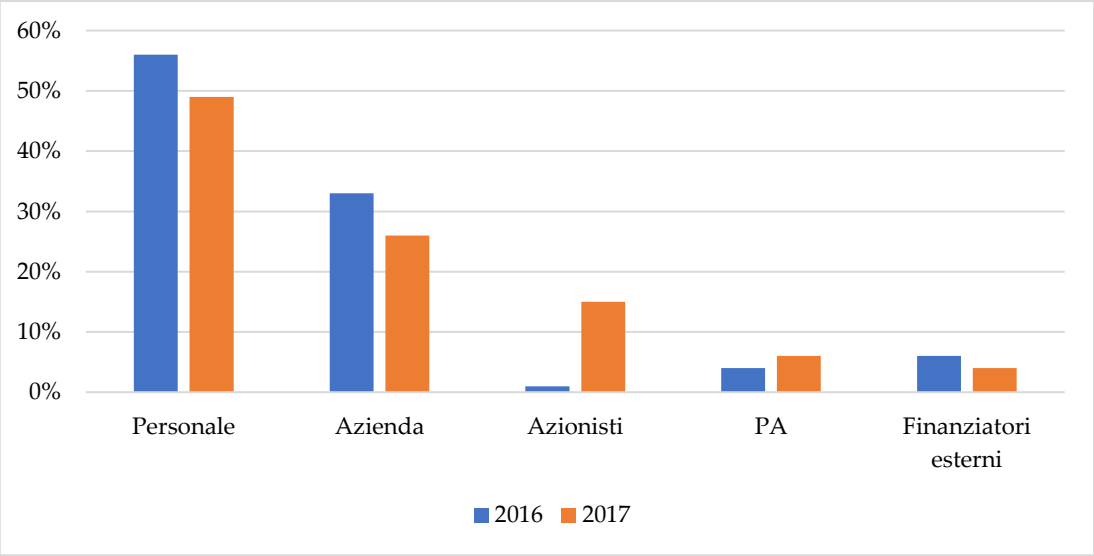
L'Italia ha visto migliorare significativamente il fatturato dei due settori presi in esame dopo quattro anni caratterizzati da una forte contrazione dei ricavi e con tassi di crescita costantemente negativi. Nominalmente, il fatturato del settore siderurgico nel 2018 si attesta sui 42,5 miliardi di euro -registrando una crescita dell'8% rispetto all'anno precedente- e circa il 34% di questo è rappresentato dalle esportazioni (anch'esse in crescita del 7,5%), arrivando dunque ad un fatturato interno netto di 28 miliardi di euro (+8,2%).

All'interno della siderurgia primaria, il principale attore risulta essere il settore delle acciaierie che, con un tasso di crescita del fatturato del 18,57%, ha

[1] http://federacciai.it/rapporto-di-sostenibilita-2019/

prodotto quasi 17 miliardi di euro nel 2017. Le esportazioni giocano un ruolo fondamentale nel livello di ricavi raggiunto con una quota del 42,8% su quest'ultimo, evidenziando una maggior propensione ad esportare un materiale come l'acciaio rispetto ad altri.

Di seguito si riporta come l'industria delle acciaierie distribuisce agli stakeholder il valore aggiunto prodotto tra il 2016 e il 2017:



Mentre le quote destinate a finanziatori esterni e allo stato sono rimaste pressoché invariate da un anno all'altro, quelle distribuite al personale e all'azienda (ovvero asset da reinvestire all'interno della stessa) si sono ridotte a favore di un aumento del valore aggiunto destinato agli azionisti, il quale è passato dall'1% a un 15%, portando nuovamente ad una sostanziale redistribuzione degli utili, in calo da diversi anni.

Valutazione rischio di credito del settore

Nel 2019 le condizioni di business per il settore siderurgico italiano hanno iniziato a deteriorarsi, dopo aver attraversato una fase positiva durante i tre

anni precedenti. Il motivo di questo rallentamento è da attribuirsi, oltre ai motivi di natura internazionale citati precedentemente, ad alcuni importanti fattori come la contrazione della domanda dei metalli -in Italia dovuta principalmente alla flessione dell'automotive- e di un'industria edile ancora in difficoltà; le conseguenze sono state una forte riduzione della quantità di acciaio prodotta di oltre il 10% nell'ultimo anno, una flessione di circa il 4% della siderurgia in termini di crescita del valore aggiunto e margini di profitto in continuo deterioramento.

Valutazione Rischio di Credito	Forte Miglioramento	Miglioramento	Stabile	Deterioramento	Forte deterioramento
Tendenza nei mancati pagamenti degli ultimi 6 mesi			✔		
Evoluzione nei mancati pagamenti per i prossimi 6 mesi				✔	
Tendenza nei fallimenti degli ultimi 6 mesi			✔		
Evoluzione nei fallimenti per i prossimi 6 mesi				✔	
Condizioni di Finanziamento	**Molto alto**	**Alto**	**Medio**	**Basso**	**Molto basso**
Livello di dipendenza dal sistema bancario		✔			
Indebitamento complessivo del settore		✔			
Livello di disponibilità delle banche a fornire credito a questo settore			✔		
Situazione Settore	**Forte Miglioramento**	**Miglioramento**	**Stabile**	**Deterioramento**	**Forte deterioramento**
Margini di profitto: tendenza degli ultimi 12 mesi				✔	
Situazione generale della domanda (vendite)				✔	

Nonostante le insolvenze all'interno del settore non siano aumentate e complessivamente il trend dei pagamenti sia in buono stato, secondo i dati e previsioni del Gruppo Atradius, si stima un aumento nei ritardi di pagamento e un numero di default aziendali in netta crescita nell'ultimo trimestre del 2019

e nel primo semestre del 2020 di circa il 4%. Se mediamente la durata media dei pagamenti era di 105 giorni, nel nuovo scenario (in cui quest'ultima è in aumento), è ragionevole aspettarsi che il flusso di cassa delle imprese sarà inevitabilmente soggetto a una forte pressione per mancanza di liquidità. I casi più gravi saranno rappresentati da quelle piccole/medie imprese con un alto grado di indebitamento e margini ridotti che, non svolgendo attività di esportazione, sono vincolate solamente alla produzione nel paese e che quindi hanno una stretta dipendenza con l'andamento della domanda italiana delle industrie "clienti".

RISCHIO DI CREDITO: REGRESSIONE LOGISTICA

Accenni sul Rischio di Credito

Il rischio di credito rappresenta la possibilità che si verifichi una perdita inattesa qualora una controparte di un'operazione, nei confronti della quale esiste un'esposizione, non adempia ai propri obblighi finanziari in merito al credito. Esso è di fondamentale importanza al fine di quantificare il VaR (Value at Risk) creditizio ovvero l'entità del capitale "di garanzia" da allocare per far fronte alle variazioni inattese che compromettono il valore economico del credito.

Per lo sviluppo successivo dei modelli di credit scoring si è utilizzato il rischio di credito nella sua accezione più stringente quindi considerando come evento creditizio il solo default/insolvenza della controparte, modellato secondo la distribuzione bernoulliana. In questo tipo di analisi non si può mai ricorrere alla distribuzione normale essendo quella delle perdite inattese inevitabilmente asimmetrica. Il default è l'evento più agevole da analizzare in Italia in quanto il credito in un paese bank based risulta essere necessariamente non quotato e, dunque, non osservabile il relativo valore di mercato.

La perdita inattesa misura la variabilità della perdita attesa nell'intorno del suo valore medio. Al contrario, la perdita attesa è allocata direttamente in conto economico ed è una funzione del valore della posizione esposta al default, la quota della perdita non recuperabile (Loss Given Default) e la probabilità di default. Quest'ultima è stata oggetto di numerosi studi e affrontata con diversi approcci dalla metà del secolo scorso.

Durante gli anni 60', gli analisti finanziari giunsero alla conclusione che c'era un chiaro problema di insufficienza di dati qualora si dovessero utilizzare le serie storiche sui dati d'impresa per valutare il rischio di credito; infatti, le imprese, anche nel caso in cui mantenessero la stessa ragione sociale, mutano nel corso degli anni rendendo la serie storica continua ma sola a livello formale, e non sostanziale. È per questo motivo che nel 1966 Beaver sviluppò un nuovo approccio mediante l'utilizzo delle Cross Section; l'idea è quella di predisporre due campioni di società "matchandole" per uno stesso settore, classe dimensionale, area geografica e orizzonte temporale: il primo composto da aziende sane -che non hanno mostrato fenomeni di default su un certo lasso di tempo- e l'altro da aziende anomale, tenendo fisso il periodo temporale. Successivamente si analizza l'andamento della media, all'interno dei due cluster, degli indicatori di bilancio che esplicitano maggiormente la condizione aziendale; infine, per valutare il rischio di credito su una nuova società si andrà a vedere se il suo indicatore è più vicino alla media di quello delle società sane o a quello delle anomale. L'analisi univariata di Beaver, ovvero considerando un indicatore alla volta, rappresentò la base per gli studi futuri come la regressione multipla e analisi discriminante.

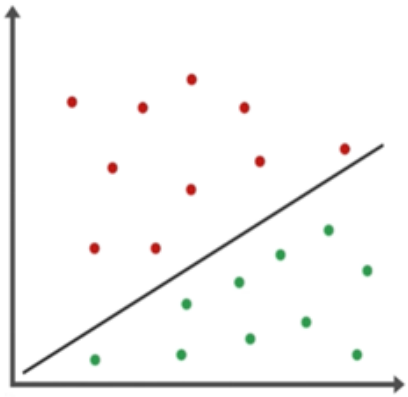
Problemi di Classificazione

La mole di dati utilizzata è esponenzialmente in crescita dagli inizi degli anni 2000. Con lo sdoganamento delle tecnologie Big Data, le grandi aziende internazionali quotidianamente raccolgono, gestiscono ed elaborano grandi quantità di dati al fine di estrarre informazioni utili per il loro business. La Data analysis varia da azienda ad azienda, di conseguenza sono stati progettati diversi modelli di dati per soddisfare le specifiche esigenze; in

particolare, la modellazione predittiva riguarda la parte della Data analysis che utilizza il Data mining e la probabilità di prevedere i risultati. In altri termini, è una tecnica statistica che prende in input dati passati e recenti, li analizza e produce un modello per identificarne il comportamento futuro. Ogni modello è caratterizzato da una o più variabili ("predittori"); una volta ricevuti i dati per ciascun predittore, l'algoritmo sviluppato potrà applicare una semplice equazione lineare oppure una complessa struttura neurale coinvolgendo le tecniche del Machine Learning e dell'intelligenza artificiale.

In questo lavoro di tesi, i modelli predittivi sono stati utilizzati per affrontare un particolare *problema di classificazione*: individuare attraverso delle predizioni se una società in un certo anno andrà o meno in stato di insolvenza. In questa tipologia di problemi, le variabili possono essere categorizzate come quantitative o qualitative. Mentre le prime assumono valori numerici (prezzo di un'azione, reddito, valore di immobili), le qualitative rappresentano, al contrario, una categoria di valori; ad esempio, possono includere il sesso di una persona (maschio o femmina) oppure la categoria di appartenenza di un prodotto (A, B o C). Generalmente, è il risultato che si vuole ottenere che suggerisce la tipologia di problema che si sta affrontando; i problemi con una risposta quantitativa vengono definiti problemi di regressione, mentre quelli che implicano una risposta qualitativa sono indicati come problemi di classificazione. Dato che l'obiettivo del progetto di tesi è quello di predire il default di una società, quindi verificarne lo stato di insolvenza in un certo anno, il problema risulta chiaramente essere di classificazione binaria (anomala/sana). Esistono diversi modelli predittivi per questo e successivamente verranno illustrati quelli utilizzati per il caso studio.

Un generale problema di classificazione si presenta nel seguente modo:



Si immagina di rappresentare le osservazioni su un grafico che ha per assi le proprietà X_i (o features) e di identificare la classe di appartenenza con un colore. Lo scopo di un modello di classificazione è quello di riuscire a trovare una funzione che meglio riesce a separare le due classi; nel caso di una regressione logistica tale funzione è rappresentata da una retta conosciuta come confine di decisione (Decision Boundary).

La retta è identificata dalla seguente equazione:

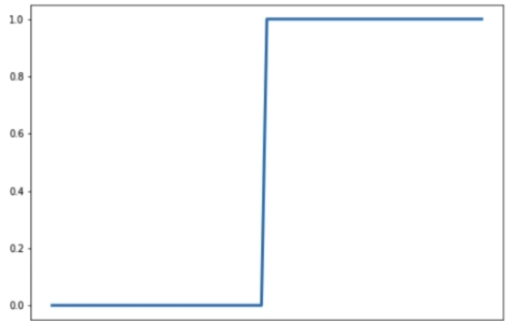
$$\alpha + \beta_1 X_1 + \beta_2 X_2 = 0$$

dove α, $\beta_{1,2}$ e $X_{1,2}$ sono rispettivamente bias, pesi del modello e proprietà. Una volta trovati bias e pesi validi per il modello durante la fase di addestramento si prosegue con la predizione trovando un certo valore z (Net input), compreso tra -∞ e +∞:

$$z = \alpha + \beta_1 X_1 + \beta_2 X_2$$

che avrà un valore ≥ 0 se il punto che si è provato a classificare si trova al di sopra della retta, mentre sarà < 0 se al di sotto. Al fine di indicare in quale delle due classi ricade il punto, si deve convertire la variabile continua z in una categorica; per ottenerla si applica una "funzione di attivazione" che prende il valore z -output del modello- e restituisce la classe corrispondente.

La funzione di attivazione più semplice è la Step Function (Funzione a Gradino):



$$\phi(z) = \begin{cases} 1 & se \quad z \geq 0 \\ 0 & se \quad z < 0 \end{cases}$$

Essa ritorna 1 (classe positiva) se $z \geq 0$ oppure 0 (classe negativa) se $z < 0$.

Tuttavia, questo modello risulta fallace sotto un aspetto fondamentale; i punti che graficamente sono più vicini alla retta hanno un grado di incertezza, relativa alla loro appartenenza alla classe, di gran lunga maggiore rispetto ai punti più distanti. Mentre la variabile continua z contiene tale informazione, con l'applicazione della funzione di attivazione vista in precedenza questa viene inevitabilmente persa avendo ottenuto come risposta solamente valori 0 o 1.

Per sapere che probabilità ha ognuno dei punti di essere classificato correttamente si deve ricorrere al modello di *Regressione Logistica*.

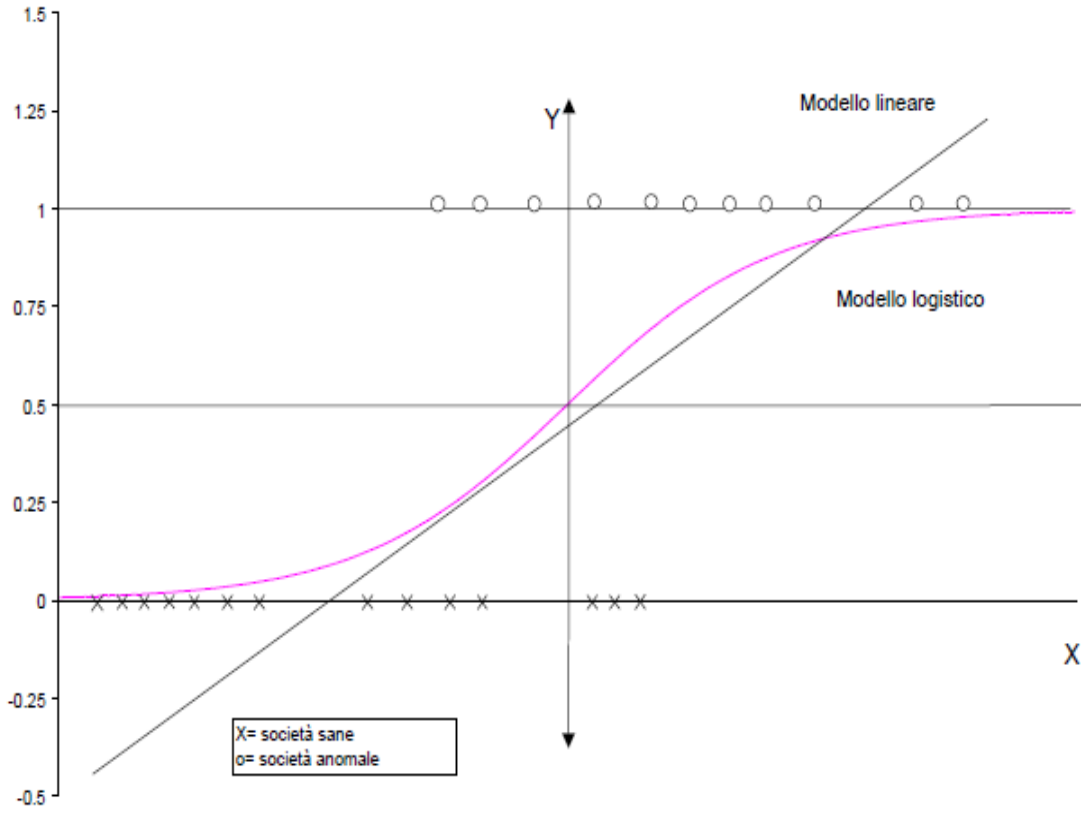
Regressione Logistica binaria

La Regressione Logistica ("logit") è uno dei modelli statistici maggiormente utilizzato nei problemi di classificazione nel Machine Learning. Come espresso in precedenza, in ogni problema di classificazione binaria l'output, chiamato "target", deve essere necessariamente una variabile categorica e il modello di regressione logistica, invece che modellarlo direttamente, fornisce la probabilità p che esso appartenga a una determinata classe. Per ottenere

questo risultato il modello logit utilizza come funzione di attivazione la "sigmoidale":

$$\boldsymbol{p(X) = \frac{1}{1+e^{-(\alpha+\beta_1 X_1+\beta_2 X_2)}}} \qquad \textbf{(1)}$$

dove $\alpha + \beta_1 X_1 + \beta_2 X_2$ è il valore continuo reale z (logit). La funzione sigmoidea ha la proprietà di ridurre i numeri reali in un range che va da 0 a 1, ovvero rappresenta la probabilità che un'osservazione ricada nella classe anomala (classe positiva, y = 1):



come si vede dal grafico, il valore della funzione sigmoidale per z = 0 corrisponde a 0,5, il quale risulta essere un perfetta soglia decisionale ("threshold") per un modello di classificazione; al di sotto di tale valore l'esempio apparterrà alla classe negativa (classe 0) e se al di sopra alla classe positiva (classe 1).

All'atto pratico, la regressione logistica non è altro che una regressione lineare alla quale si applica una specifica funzione di attivazione sigmoidea per eseguire una classificazione.

Per quanto concerne i coefficienti α e β, i dati di addestramento -chiamati training data- permettono la loro stima; mentre nel modello di regressione lineare viene utilizzato l'approccio dei minimi quadrati, nella regressione logistica è di uso comune il metodo della Massima Verosimiglianza:

$$\boldsymbol{L}\,(\boldsymbol{\alpha},\boldsymbol{\beta}) = \prod_{\boldsymbol{y_i}=\mathbf{1}} \boldsymbol{p}(\boldsymbol{X_i}) \prod_{\boldsymbol{y_i'}=\mathbf{0}} (\mathbf{1}-\boldsymbol{p}(\boldsymbol{X_i'}))$$

$$\boldsymbol{L}\,(\boldsymbol{\alpha},\boldsymbol{\beta}) = \prod_{\boldsymbol{y_i}=\mathbf{1}} \frac{\mathbf{1}}{[\mathbf{1}+\ \boldsymbol{e}^{-(\boldsymbol{\alpha}+\boldsymbol{\beta}\boldsymbol{X_i})}]} \prod_{\boldsymbol{y_i'}=\mathbf{0}} \frac{\mathbf{1}}{[\mathbf{1}+\ \boldsymbol{e}^{-(\boldsymbol{\alpha}+\boldsymbol{\beta}\boldsymbol{X_i'})}]}$$

Per agevolare i calcoli al computer, durante la fase di addestramento è consuetudine utilizzare il logaritmo della massima verosimiglianza:

$$\boldsymbol{log}\ \boldsymbol{L}\,(\boldsymbol{\alpha},\boldsymbol{\beta}) = \sum [\boldsymbol{y_i}\ \boldsymbol{log}(\boldsymbol{p}(\boldsymbol{X_i})) + (\mathbf{1}-\boldsymbol{y_i})\ \boldsymbol{log}(\mathbf{1}-\boldsymbol{p}(\boldsymbol{X_i}))]$$

Con questo metodo iterativo si cercano α e β_i tali che la probabilità che il punto osservato sia nella classe anomala, prevista nella formula (1), corrisponda il più fedelmente possibile al reale stato del punto stesso. In particolare, si stimano tutti i coefficienti in modo tale che, inserendoli nel modello p(X), si ottenga come risposta un valore vicino a 1 per tutte quelle osservazioni anomale e vicino a 0 viceversa andando. La finalità della fase di addestramento è quella di massimizzare la funzione *log L(α,β)* tramite un algoritmo di ottimizzazione chiamato Gradient Ascent che, appunto, ne ricerca il punto di massimo. Un'alternativa è rappresentata dal minimizzare la funzione di costo *Log Loss* abbinata al metodo di ottimizzazione Gradient Descent il quale ne individua il punto di minimo; quest'ultimi concetti verranno approfonditi in seguito.

La regressione logistica è una tecnica molto utilizzata nel momento in cui si desidera modellare i dati binari e, soprattutto, quando vi è la necessità di dare una precisa interpretazione (ad esempio economica/finanziaria) al modello. Di contro, questa non risulta essere la soluzione ideale qualora i dati non fossero separabili linearmente e quando l'obiettivo principale, invece che essere l'interpretazione, è la performance; infatti, si tratta di un modello relativamente semplice che, in generale, offre prestazioni inferiori rispetto a strutture più complesse come le reti neurali.

RETE NEURALE ARTIFICIALE

Dall'AI al Deep Learning

L'intelligenza Artificiale (AI) è la branca dell'informatica che si occupa di fornire ai computer l'abilità di risolvere quei problemi che tipicamente richiedono l'utilizzo dell'intelligenza. Esempi di questi problemi sono la comprensione della lingua parlata, riconoscimento di immagini e scrittura a mano o il ragionamento strategico. Per affrontare queste attività apparentemente semplici, gli esseri umani delegano tutto il lavoro alla più potente macchina di calcolo distribuito, chiamata cervello, la quale utilizza la conoscenza pregressa e immaginata per risolvere problemi.

Dagli anni 70' vari ricercatori hanno provato a codificare tutta l'informazione possibile all'interno dei computer al fine di creare una macchina intelligente ma si resero conto che la complessità in gioco era troppo elevata. Negli anni successivi un secondo approccio, che inizialmente veniva sottovalutato, si affermò definitivamente: invece che immettere manualmente l'informazione, si pensò a delle tecniche per insegnare ai computer come apprendere autonomamente dai dati, ovvero quello che si intende oggi per Machine Learning (ML).

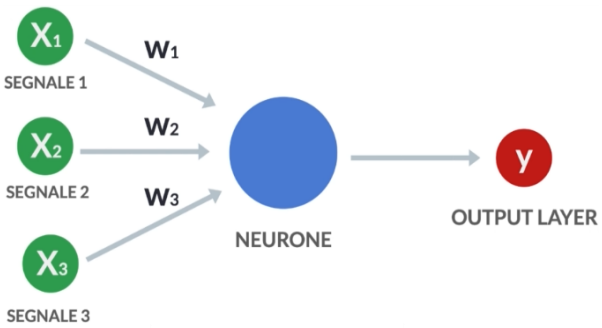
Fu un passo tanto importante quanto inevitabile dato che per alcuni problemi (come insegnare a guidare ad un'automobile) il numero di istruzioni da inserire nel potenziale algoritmo sarebbe stato enorme; il ML evita tutto questo tramite un processo di addestramento nel quale cerca di individuare delle relazioni (pattern) all'interno dei dati che ha in ingresso dando di conseguenza la capacità al computer di autoprogrammarsi.

Il problema fondamentale degli algoritmi di ML risiede nel fatto che non riescono a rilevare pattern non lineari all'interno dei dati; questa forte limitazione viene sorpassata dall'avvento del Deep Learning che nasce al fine di realizzare modelli più robusti e in grado di individuare anche relazioni non lineari tra i dati. I modelli in questione sono le Reti Neurali Artificiali Profonde (RNA) che ad oggi stanno rivoluzionando l'intero settore dell'AI.

Funzionamento di una RNA

Le RNA sono la normale rappresentazione del cervello: ogni qual volta si riceva uno stimolo nervoso, alcuni dei centinaia di miliardi di neuroni si attivano e rilasciano un segnale di attivazione che si propaga a cascata ai neuroni vicini; quelli che si attivano ripetutamente nello stesso istante si legheranno formando complesse strutture chiamate reti neurali. Questa è la base dell'apprendimento negli umani e negli animali.

Il grafico seguente rappresenta un neurone artificiale come se fosse un modello di ML:



si ha un certo numero di segnali X_i, provenienti da altri neuroni o altri processi, che raggiungono il neurone grazie a delle connessioni ponderate (come se fossero delle sinapsi), ognuna delle quali ha una propria intensità, o peso, w_i.

All'interno del neurone i segnali vengono moltiplicati per il rispettivo peso e sommati ottenendo il "potenziale d'azione":

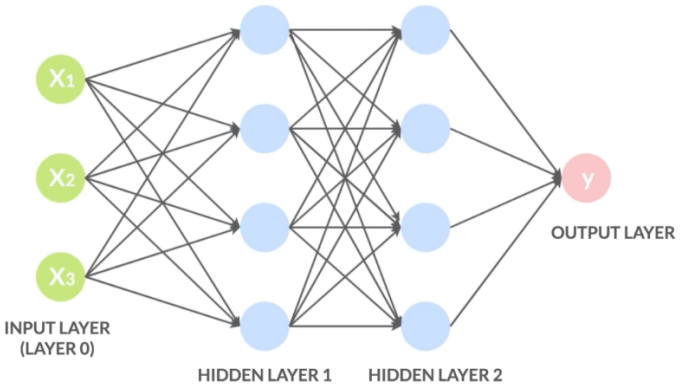
$$z = w_1X_1 + w_2X_2 + w_3X_3$$

Successivamente z passa attraverso una precisa funzione di attivazione che avrà il compito di stabilire se è sufficientemente intenso da poter essere propagato al neurone vicino per attivarlo:

$$y = \phi(z)$$

dove y rappresenta l'output del neurone. Aggiungendo il bias, la struttura presente in figura ricalca quella dei modelli di regressione; se lineare o logistica dipende unicamente dalla tipologia di funzione di attivazione utilizzata (ad esempio con la sigmoidale si avrà una regressione logistica come visto in precedenza).

Una Rete Neurale Artificiale trae ispirazione dal processo di apprendimento che si verifica nel cervello biologico disponendo svariate unità di calcolo su più livelli o strati in modo tale che essa raggiunga un grado di astrazione sempre maggiore:



in figura sono rappresentati tre neuroni -features iniziali- nel primo strato di input, quattro neuroni in ciascuno dei due strati nascosti successivi e uno strato di output. Una RNA con due o più strati nascosti è definita "profonda". I nodi presenti nel secondo strato nascosto sono generati combinando i nodi del primo hidden layer, i quali sono a loro volta un'astrazione dei nodi di input. Gli strati nascosti contengono nodi che non necessariamente hanno un senso logico e di conseguenza non sono facilmente interpretabili; in ognuno di questi la rete utilizza le informazioni provenienti dallo strato precedente per

apprenderne delle nuove maggiormente significative al fine di risolvere un determinato problema. Il primo strato prende in input i dati e, dopo averli processati, produce un output che corrisponde esattamente all'input dello strato successivo; quando, come in questo caso, ogni neurone di uno strato precedente è collegato ad ogni neurone di quello successivo, la rete viene definita "densa". Il numero degli strati nascosti e dei nodi in essi presenti (che possono variare tra i diversi strati) sono iperparametri che devono essere definiti a priori e che determinano la complessità e la potenza della rete. La sfida, dunque, è quella di trovare la soluzione migliore che permette di ottimizzare i vari iperparametri nel minor tempo e consumando la minor quantità di risorse possibile.

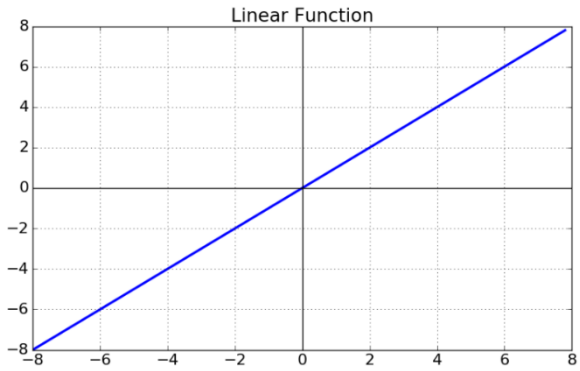
Tipologie di Funzioni di Attivazione

Le funzioni di attivazione (AF) sono equazioni matematiche indispensabili nello sviluppo di una generica RNA in quanto determinano l'output di ogni neurone in ognuno degli strati; esse fungono da "cancello matematico" tra l'input che arriva al neurone e la sua risposta diretta allo strato successivo. Importanti parametri come l'accuratezza/efficienza computazionale dell'addestramento di un modello e la capacità/velocità della rete di convergere sono fortemente influenzati dalla tipologia di funzione di attivazione scelta dall'analista.

Lo scopo principale delle AF è introdurre proprietà non lineari nella rete neurale; infatti, se queste non venissero applicate il segnale in uscita di un qualunque nodo sarebbe una funzione lineare, in particolare un polinomio di grado 1, che non ha un sufficiente livello di complessità per apprendere l'architettura dei dati e fornire previsioni accurate. Dunque, in presenza di

attivatori lineari una rete neurale corrisponderebbe esattamente a un modello di regressione lineare con capacità e potenza molto limitate.

Come si vede dal grafico riportato di seguito, la funzione di attivazione lineare o Identità produce una risposta non confinata all'interno di un range:



$$\phi(z) = z$$

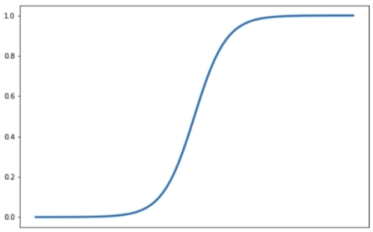
Prende gli input, moltiplicati per i pesi, e crea un segnale di output proporzionale all'input stesso.

Un attivatore lineare di questo tipo non può essere considerato migliore di uno a Gradino (descritto precedentemente) solamente perché, avendo un range da -∞ a +∞, consente più output in quanto presenta due gravi problemi: il primo è legato all'impossibilità di utilizzare la backpropagation (Gradient Descent) - che verrà descritta nel capitolo successivo- per addestrare il modello dato che, avendo costante la derivata della funzione, questa non può avere alcuna relazione con l'input; quindi non è possibile "tornare indietro" per capire quali pesi possono offrire una previsione migliore. L'altro lato negativo riguarda il fatto che, qualunque sia il numero di strati, utilizzando un attivatore lineare il livello di output sarà inevitabilmente una combinazione lineare del primo strato e, dunque, una AF lineare è come se riducesse l'intera rete neurale in un unico strato.

Mentre le FA lineari non hanno un reale utilizzo pratico, quelle non lineari sono alla base di ogni tipologia di modello del Machine Learning in quanto

consentono la Backpropagation -avendo una funzione derivata correlata agli input- e, inoltre, permettono la realizzazione di una rete neurale profonda composta da più strati nascosti volta a individuare le complesse mappature tra input e output; tra le più importanti ci sono:

- Sigmoide:

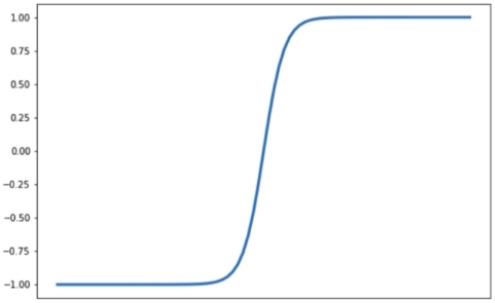


$$\phi(z) = \frac{1}{1 + e^{-z}}$$

è un AF non lineare che viene utilizzata al fine di prevedere gli output sulla base della probabilità in quanto permette di comprimere ogni valore reale in un range compreso tra 0 e 1.

È presente principalmente nei modelli di regressione logistica e nello strato di output dei modelli di deep learning volti alla risoluzione dei problemi di classificazione binaria.

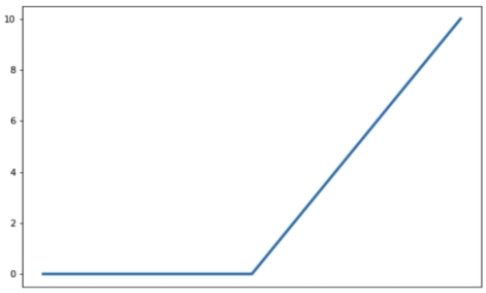
- Tangente iperbolica (tanh):



$$\phi(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$

la Tangente iperbolica risulta molto simile alla sigmoidea in quanto ne rappresenta una versione scalata. È centrata sullo 0 con un range compreso tra ±1; questa caratteristica è il suo più grande vantaggio dato che così supporta il processo di backpropagation. La funzione tanh è stata utilizzata principalmente nelle reti neurali ricorrenti per l'elaborazione del linguaggio naturale e le attività di riconoscimento vocale.

- Rectified Linear Unit (ReLU):

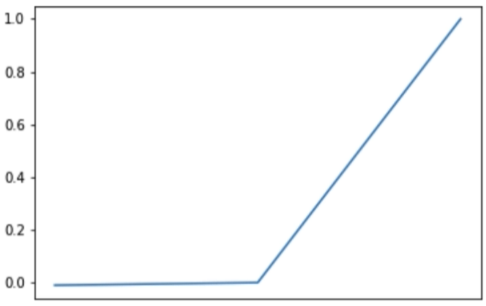


$$\phi(z) = max(0, z)$$

questa ritorna 0 per valori che sono ≤ 0 oppure il valore stesso per valori > 0. Rispetto ad altre AF, come Sigmoide e tanh, offre prestazioni e una generalizzazione molto migliori nel deep learning; Il vantaggio maggiormente significativo nell'utilizzare la ReLU risiede nella velocità del calcolo e nell'efficienza computazionale: eseguendo semplicemente un'operazione di soglia su ogni elemento di input in cui tutti i valori inferiori o uguali a zero sono impostati a zero, essa non necessita di particolari calcoli e converge molto rapidamente.

Quando gli input sono negativi oppure si trovano nell'intorno dello zero, il gradiente della funzione diventa 0 eliminando la possibilità di eseguire la backpropagation e di continuare ad apprendere; tale problema, chiamato "Dying ReLU", è però facilmente superabile utilizzando una variante della ReLU:

- Leaky ReLU:



$$\phi(z) = max(0.01z, z)$$

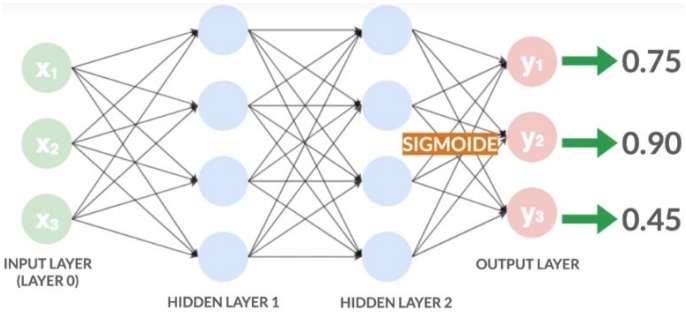
piuttosto che assegnare 0 a tutti i valori di input ≤ 0, ritorna un valore molto piccolo (generalmente settato a 0,01*z) ma crescente; fornendo una pendenza

dalla parte negativa della funzione, si rende possibile la backpropagation e la possibilità di apprendere il valore più appropriato di z.

- Softmax:

Le funzioni di attivazione sigmoidale e tanh sono utilizzate per i problemi di classificazione binaria nei quali le classi da distinguere sono soltanto due; questi vengono trattati, come esposto precedentemente, inserendo un unico nodo posto nello strato di output che indica se l'osservazione in questione appartiene o meno alla classe positiva (anomala).

Esistono però altri problemi in cui le classi da individuare sono più di due: in questi casi si possono avere due diversi tipi di problema; nel primo le classi sono indipendenti tra di loro e quindi una stessa osservazione può appartenere a più di una classe (l'appartenenza ad una classe non impedisce la possibilità di appartenenza ad un'altra). Fa parte di questo primo caso il problema di stabilire con una rete neurale di quali argomenti tratta un articolo di giornale ("uno sportivo molto famoso e una diva della TV stanno lavorando ad uno spot per la promozione del nuovo iPhone"); in questo esempio l'articolo appartiene contemporaneamente a tre categorie, anche se non hanno la stessa rilevanza all'interno dello stesso: sport, attualità e tecnologia.



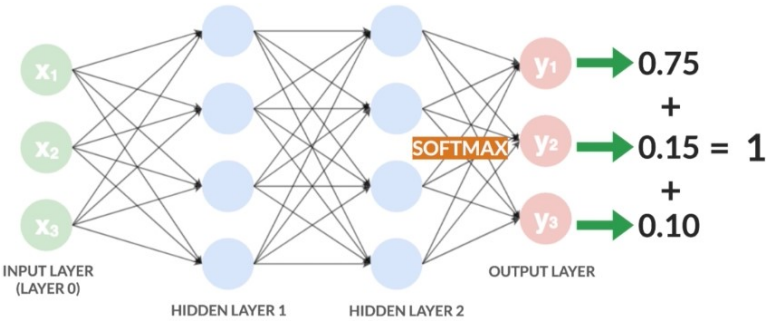
I problemi della prima tipologia vengono affrontati ponendo un numero di nodi di output pari al numero di classi da individuare. Dato che si utilizza la funzione Sigmoide come attivatore dell'ultimo strato, le risposte, indipendenti tra loro, saranno sempre tra 0 e 1 e ognuna delle quali rappresenterà la probabilità che l'osservazione appartenga

alla classe in questione. Questo procedimento equivale a prendere gli output dell'ultimo strato nascosto e utilizzarli per realizzare un modello di regressione logistica per ognuna delle classi.

Il secondo caso riguarda le classi esclusive, ovvero quando l'osservazione può appartenere ad un'unica classe, eliminando la possibilità che appartenga alle altre. L'esempio tipico è rappresentato dal riconoscere una tipologia di animale (cane, gatto, topo) nel quale la classe di appartenenza corretta sarà sempre e solo una. È per questi problemi che si utilizza la funzione di attivazione Softmax:

$$\phi(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_K}} \qquad \textit{per } j = 1, \ldots, K$$

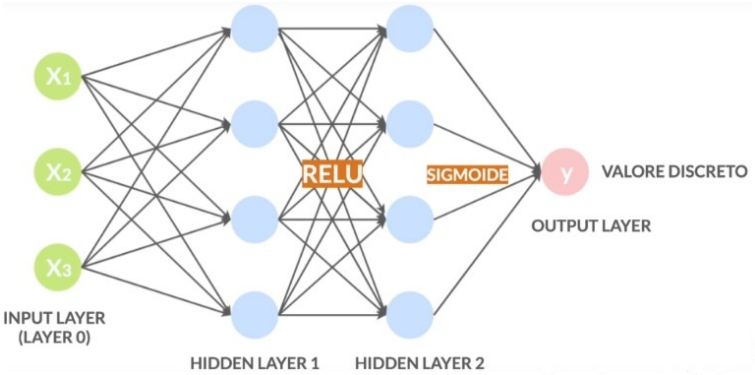
Essa ci permette di generalizzare la regressione logistica a problemi di classificazione multiclasse:



utilizzando la Softmax si ottiene un numero di output sempre uguale al numero di classi del dataset; tuttavia, questa volta saranno dipendenti tra loro in quanto la somma degli output è normalizzata a 1 e il valore di ognuno di essi rappresenta la probabilità che l'osservazione appartenga a quella determinata classe; la classe da prendere come corretta sarà quella con la più alta probabilità.

Risulta molto difficile individuare quale sia la combinazione di AF ottima per qualsiasi caso di studio anche perché ogni strato della rete può avere un attivatore diverso dagli altri. In generale per i problemi di classificazione binaria -oggetto del lavoro di tesi-, una buona tecnica risulta essere quella di

adottare la ReLU tra i vari strati nascosti e la Sigmoide nell'ultimo strato in modo tale da fornire un output sottoforma di probabilità:



il modello di RNA presentato in figura è chiamato Percettrone Multistrato. Nonostante la pratica suggerisca l'utilizzo della ReLU, risulta conveniente verificare anche le capacità della tangente iperbolica.

Al contrario, per i problemi di regressione la tecnica comune è la stessa ma con una funzione di attivazione lineare (Identità) nell'ultimo strato nascosto per far in modo che ritorni un valore continuo, ovvero il net input proprio dell'ultimo livello.

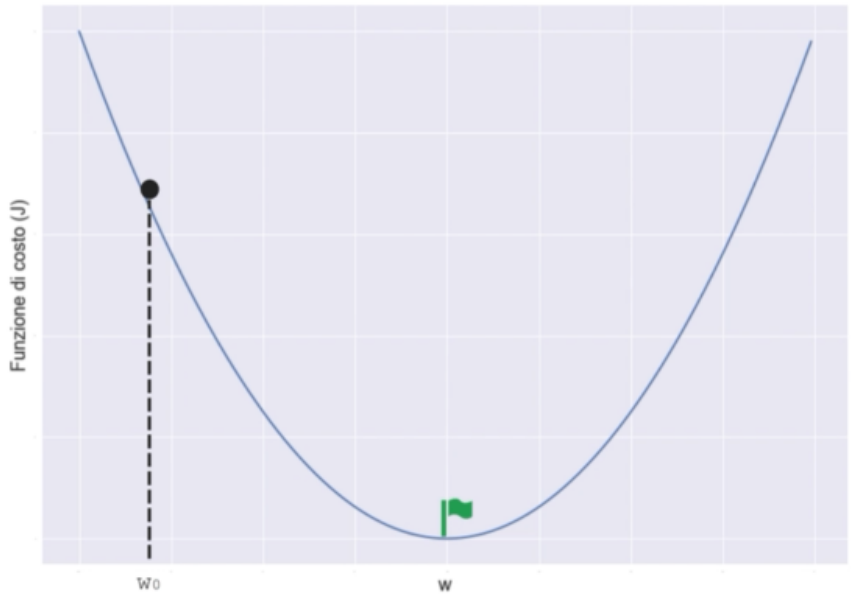
Addestramento di una RNA

Creare variabili e farle interagire tra loro è necessario, ma assolutamente non sufficiente per far sì che l'intera rete neurale apprenda da sola. Deve essere predisposta una grande mole di dati da fornire alla rete; i dati includono sia gli input sia gli output attesi dalla rete neurale. In generale, quando viene fornito un input alla rete neurale, questa restituisce un output; al primo tentativo, a meno di colpi fortunosi, la rete non può ritornare l'output corretto ed è per questo che, durante la fase di apprendimento, ogni esempio in ingesso viene inserito con la sua target che spiega quale output la rete avrebbe dovuto indovinare. Se l'output fornito della rete e la target coincidono allora i parametri del modello vengono mantenuti; in caso contrario, ovvero se

l'output ottenuto non corrisponde all'etichetta, i pesi vengono modificati. Nel corso di questo capitolo si andrà ad approfondire l'intero processo di apprendimento e addestramento di una RNA al fine di comprendere come la rete stessa arriva a determinarne la soluzione ottimale.

Gradient Descent

È l'algoritmo di ottimizzazione per eccellenza; è utilizzato per apprendere i coefficienti di un modello di ML (pesi e bias).

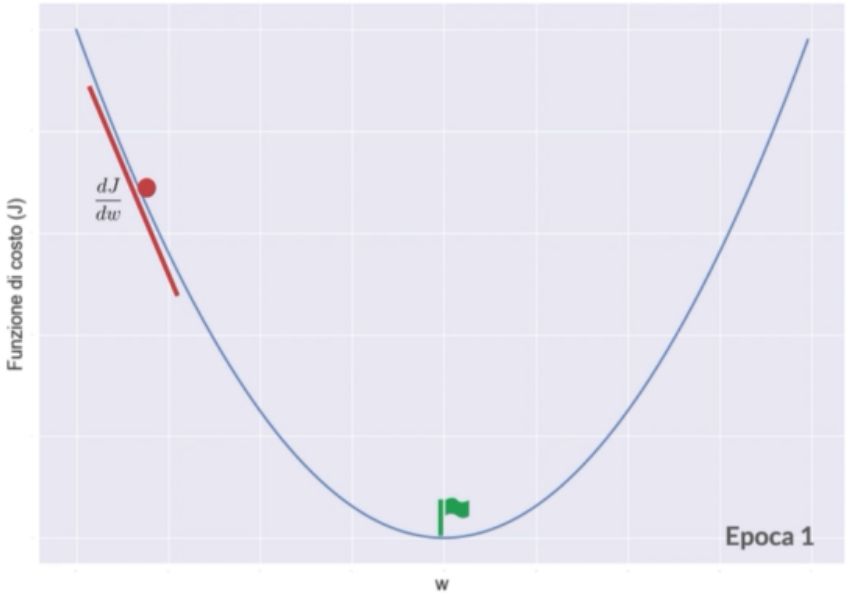


Si considera un grafico che mostra la variazione di una funzione di costo rispetto al coefficiente del modello (si supponga di averne un unico peso w). L'obiettivo del Gradient Descent è quello di trovare i coefficienti del modello che minimizzano la funzione di costo, quelli dunque che ci offrono il modello migliore.

Il primo step è quello di inizializzare il coefficiente ad un valore casuale w_0 al quale sarà associato un certo valore della funzione di costo. Successivamente si valuta la derivata della funzione di costo J(w) rispetto al coefficiente w, nel

punto w_0; tale valore ci dirà se e quanto velocemente la funzione di costo sta crescendo o decrescendo in quel preciso punto. Nel caso in figura, la pendenza è negativa, dunque il valore della derivata sarà negativo e sottraendola al valore corrente del coefficiente si registra inevitabilmente un aumento del valore di w (si sposta verso destra, verso il punto di minimo). Se al contrario la derivata fosse stata positiva allora si sarebbe ottenuta una riduzione del coefficiente.

Di seguito è illustrato il codice python del Gradient Descent:

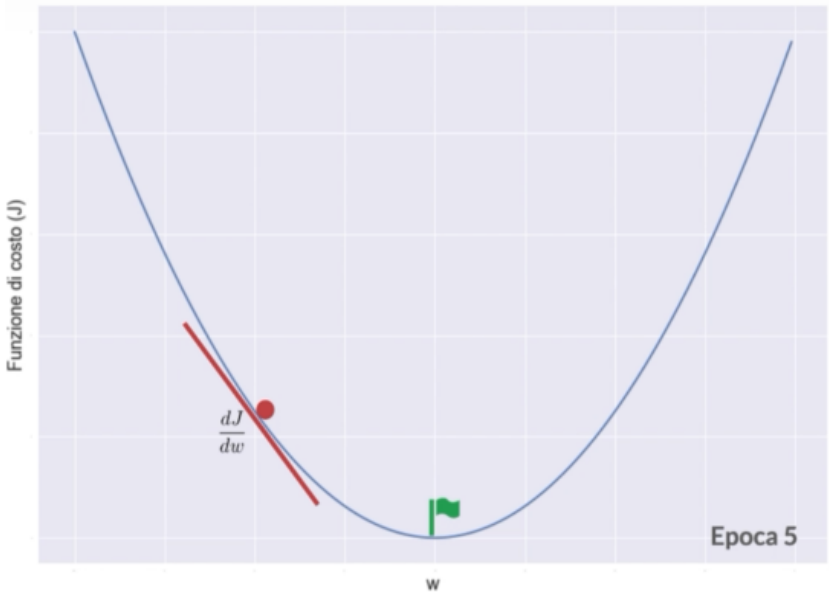



```
w = rand ()
        for epoch in epochs:
        dw = cost_derivative ()
        w = w - α*dw
return w
```

Prima di effettuare la sottrazione si deve moltiplicare la derivata per un fattore costante α chiamata *learning rate*[2] (lr) che permette di controllare la velocità di

[2] Approfondendo, il tasso di apprendimento misura quanto cambiare il modello in risposta all'errore stimato ogni volta che i pesi vengono aggiornati. Un lr più piccolo richiede un numero di *epoche* più elevato dato che le modifiche sui pesi sono più piccole; al contrario, tassi più alti si traducono in un minore numero di epoche in quanto i cambiamenti sono rapidi.

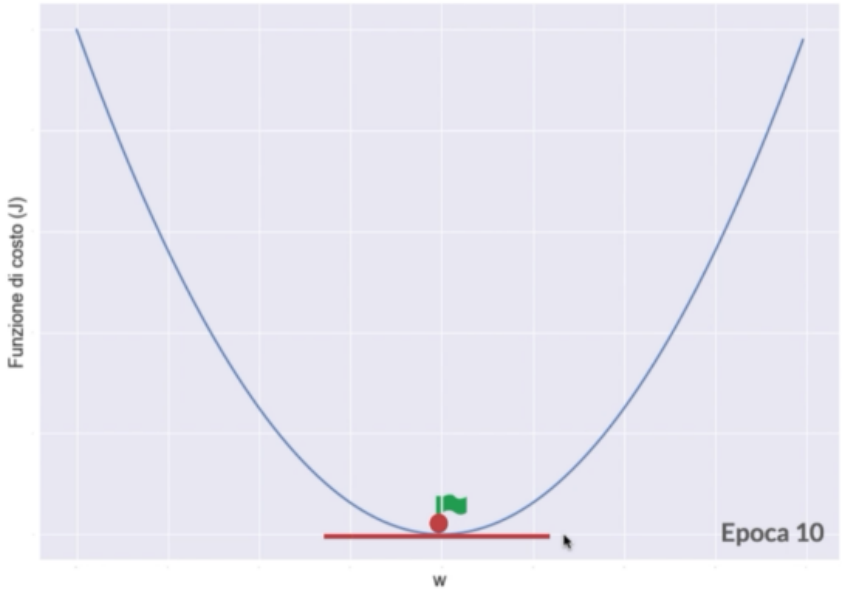
ogni step del Gradient Descent; a valori di α infinitesimali (α=0,000001) corrispondono step troppo piccoli e, conseguentemente, un tempo eccessivamente lungo per raggiungere la convergenza nel punto di minimo. Allo stesso modo, ponendo α grande (α=100) si rischia che il coefficiente w rimbalzi da una parte all'altra della curva (apprendimento instabile). Quindi, il learning rate è un altro iperparametro che si deve ottimizzare; normalmente il valore corretto è da ricercarsi in un range tra 10^{-4} e 1.



Il calcolo progressivo della derivata della funzione di costo e la sottrazione sono operazioni da ripetere per un numero predefinito di cicli chiamato *epoche*[3], le quali con il passare del tempo permetteranno al coefficiente di convergere verso il valore ottimale; man mano che il coefficiente si avvicina a tale punto la pendenza della funzione si riduce, il valore della derivata diminuisce e gli step successivi saranno sempre più brevi.

[3] Epoche: numero di volte che il Gradient Descent passa su tutto il train set.

Il tutto fino ad arrivare al punto di minimo in cui la derivata è nulla oppure al raggiungimento di un intorno di questo sufficientemente piccolo da ritornare una buona approssimazione del coefficiente:



Se si aggiungono il bias e ulteriori pesi a questo modello, l'algoritmo si conserva ma andrà a valutare il gradiente (insieme delle derivate parziali) invece che la singola derivata; attraverso il seguente programma python si può addestrare alcuni modelli di ML come una regressione lineare o logistica:

```
def gradient_descent (X, y, epochs = 100):
    w = rand (X.shape[1], 1)
    b = rand (1, 1)
    for _ in range (epochs):
        dw, db = cost_gradient (w, b, X, y)
        w = w - α*dw
        b = b - α*db
    return w, b
```

L'ultimo elemento ancora rimasto in sospeso è il calcolo del gradiente della funzione di costo; di seguito sono rappresentate rispettivamente la funzione di costo per una regressione lineare, ovvero la somma dei quadrati residui, e per una regressione logistica che in questo caso è una Log Loss, dato che viene utilizzato come ottimizzatore il Gradient Descent:

$$J(w,b) = \sum_{i=1}^{N} (y_i - (w_i X_i + b))^2$$

$$J(w,b) = -\frac{1}{N}\sum_{i=1}^{N} [y_i \, log(\phi(w_i X_i + b) + (1 - y_i) \, log(1 - \phi(w_i X_i + b)]$$

dove $\phi(w_i X_i + b) = \phi(z) = p(X_i)$, quindi la probabilità che l'osservazione ricada nella classe anomala (classe positiva, y = 1).

Differenziando le funzioni di costo rispetto al coefficiente w e bias b si ottengono le stesse derivate parziali sia nel caso lineare sia in quello logistico, le quali andranno a comporre il gradiente della funzione di costo rispetto ai coefficienti (w,b):

$$\nabla J(w,b) = [\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}]$$

Infine, si mostra un'implementazione in python del Gradient Descent con i passaggi necessari per effettuare le derivate parziali della funzione di costo:

```
def gradient_descent (X, y, α = 0.001, epochs = 100):
    w = rand (X.shape[1], 1)
    b = 0
    for _ in range (epochs):
        z = dot (X,w) + b
        a = activation (z)
        dw = X.T.dot (a-y)
        db = sum (a-y)
        w = w - α*dw
        b = b - α*db
    return w, b
```

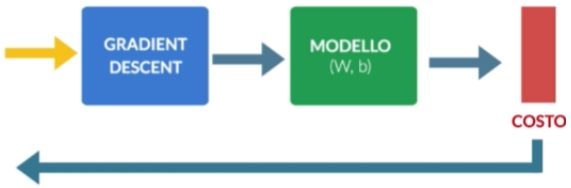
Prima si calcolano gli output lineari moltiplicando i pesi w per le features X e aggiungendo il bias b; lo si passa alla funzione di attivazione che fornirà la risposta effettiva del modello e successivamente si applicano le due learning rules del Gradient Descent per calcolare le derivate parziali rispetto ai pesi e bias; infine, si aggiornano pesi e bias sottraendo il prodotto tra derivate

parziali e il learning rate. Di seguito vengono esposte le tre principali varianti del Gradient Descent.

Full Batch Gradient Descent

Il FBGD ha un'implementazione identica a quella del Gradient Descent precedentemente descritto; esegue il GD utilizzando tutti gli esempi del set di addestramento per ogni singolo step dell'algoritmo.

	X1	X2
es1	…	…
es2	…	…
…	…	…
…	…	…
esN-1	…	…
esN	…	…

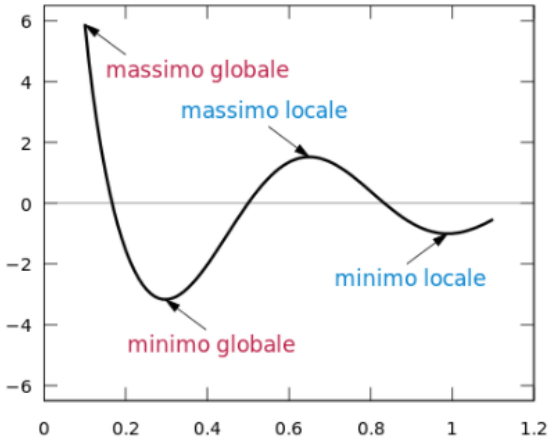


Si selezionano tutte le osservazioni del training set e si esegue un primo step del GD: si calcola il gradiente della funzione di costo e, utilizzando tali derivate parziali, vengono aggiornati pesi e bias del modello, il quale potrà essere utilizzato per valutare il valore della funzione di costo; essendo al termine della prima epoca di addestramento, la funzione di costo avrà un valore molto alto.

La seconda epoca ripercorre nuovamente le operazioni sopra descritte: si effettua il primo step del GD, vengono aggiornati i valori di pesi e bias e calcolata la funzione di costo che verosimilmente sarà di entità minore rispetto alla precedente. Si ripetono ciclicamente gli stessi passi per un determinato

numero di epoche al fine di aggiornare i valori di pesi e bias e ottenere il valore della funzione di costo più basso possibile (ottimale).

Il FBGD presenta dei limiti importanti legati all'inefficienza per dataset di grandi dimensioni (Gigabyte), in quanto si dovrebbe caricare l'intero dataset in memoria, e alla poca dinamicità dato che per migliorare il modello con nuovi dati è necessario riaddestrarlo sull'intero dataset. Tuttavia, il problema maggiore è rappresentato dalla presenza dei minimi locali. Le funzioni di costo raramente sono perfettamente convesse con un unico punto di minimo; nella realtà i modelli di ML più avanzati utilizzano funzioni di costo che possono avere più punti massimo/minimo:



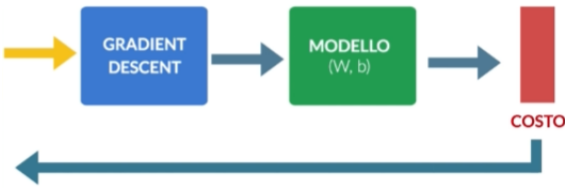
I punti di minimo locale sono tutti quei punti in cui una funzione passa dall'essere decrescente all'essere crescente e viceversa per quanto concerne i punti di massimo globale. I punti di minimo e massimo globale sono i punti in cui la funzione assume rispettivamente valore minimo e massimo.

I punti di minimo locale rappresentano un problema per il GD (o FBGD) in quanto c'è il forte rischio che questo si fermi in uno di questi punti, fornendo così una funzione di costo non ottimizzata (non corrispondente al suo valore minimo).

Stochastic Gradient Descent

Lo SGD è una variante che elimina i problemi del Gradient Descent eseguendo il GD, invece che sull'intero training set, su una singola osservazione per volta.

	X1	X2
es1	…	…
es2	…	…
…	…	…
…	…	…
esN-1	…	…
esN	…	…



Si utilizza un unico primo esempio per calcolare il gradiente della funzione di costo, aggiornare i valori di pesi e bias e valutare la funzione di costo. Successivamente si torna indietro, si mischia l'ordine delle osservazioni all'interno del dataset per evitare la formazione di cicli e, selezionando un secondo esempio, si ripete il processo; un'epoca si ritiene conclusa quando lo SGD ha processato su ogni esempio del training set. Utilizzando questo metodo di ottimizzazione è molto probabile che il valore della funzione di costo, invece di diminuire costantemente come avveniva con il FBGD, tenda inizialmente a oscillare; solamente al termine di diverse epoche si registrerà una sua significativa riduzione fino al valore ottimale.

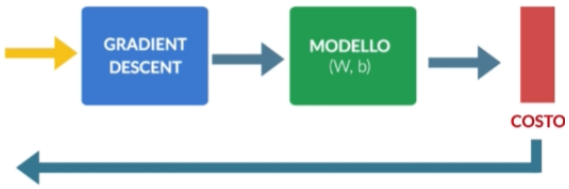
```
def stochastic_GD (X, y, epochs = 100):
    w = rand (X.shape[1], 1)
    b = rand (1, 1)
    for _ in range (epochs):
        for x_i, y_i in zip (X,y):
            z = dot (x_i,w) + b
            a = activation (z)
            dw = (a-y_i)*x_i
            db = a-y_i
            w = w - α*dw
            b = b - α*db
return w, b
```

Il codice python del SGD differisce dal FBGD per la presenza in questo caso di un secondo ciclo in cui si seleziona quell'unico esempio alla volta e lo si utilizza per eseguire uno step del GD. Lo SGD è un metodo molto dinamico in quanto per aggiornare il modello con nuovi dati è sufficiente eseguire uno step del GD solamente su tali dati e non sull'intero dataset; occupa poca memoria caricando un esempio per volta e, grazie alle oscillazioni della funzione di costo esempio dopo esempio, è generalmente meno affetto dal problema dei punti di minimo locale. Di contro ha il fatto che se le fluttuazioni fossero eccessive si rischierebbe di saltare completamente il punto di minimo globale.

Mini Batch Gradient Descent

Una soluzione intermedia tra il FBGD e lo SGD è rappresentata dal Mini Batch Gradient Descent; consiste nell'eseguire il GD su un numero determinato di osservazioni alla volta.

	X1	X2
es1	…	…
es2	…	…
…	…	…
…	…	…
esN-1	…	…
esN	…	…



Si seleziona un certo numero *s* di esempi, si esegue uno step del GD, si aggiornano i pesi e bias del modello e si valuta la funzione di costo; successivamente si selezionano altri *s* esempi del training set e ripetiamo il processo. Anche in questo caso un'epoca si conclude quando si è eseguito il

GD su tutti gli esempi. Il valore della funzione di costo oscillerà ma non eccessivamente come nel caso dello SGD.

```
def mini_batch_GD (X, y, batch_size = 32 epochs = 100):
    w = rand (X.shape[1], 1)
    b = rand (1, 1)
    batches = X.shape[0]/batch_size
    for _ in range (epochs):
        for batch in range (batches):
            X_batch = X[batch*batch_size:(batch+1)*batch_size,:]
            y_batch = y[batch*batch_size:(batch+1)*batch_size]
            z = dot (x_batch,w) + b
            a = activation (z)
            dw = X_batch.T.dot(a-y_batch)
            db = sum(a-y_batch)
            w = w - α*dw
            b = b - α*db
return w, b
```

Anche in questo caso è presente un secondo ciclo che itera sul numero di batch in cui è stato diviso il training set; questo numero si ottiene dividendo il numero totale di osservazioni per il numero *batch_size*[4] di esempi per ogni batch (in generale è consigliato avere un *batch_size* multiplo di 2). Le prime due istruzioni dopo il secondo ciclo for permettono di estrarre il batch corrente dall'intero set di addestramento.

Tale soluzione è corretta solo se il numero di esempi del set di addestramento è un multiplo del *batch_size*; in caso contrario si crea inevitabilmente un ultimo batch incompleto rispetto ai precedenti. Dunque, si procede all'inserimento di un'istruzione condizionale "*if else*" nella quale si impone che se il batch corrente è anche l'ultimo allora dovrà estrarre tutti gli esempi rimasti, altrimenti estrarrà un normale batch con un numero di esempi pari al *batch_size* definito in precedenza.

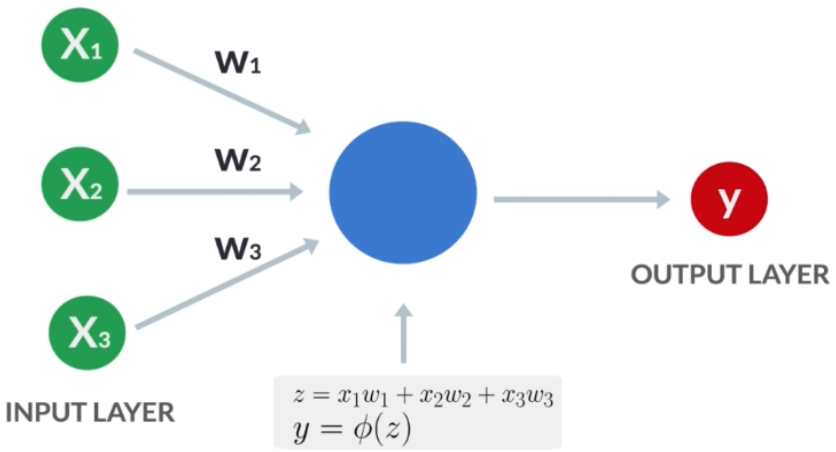
[4] batch_size: rappresenta il numero di osservazioni del set di addestramento che vengono propagate attraverso la rete. Ad esempio, se si hanno 1000 esempi nel training set e il batch_size=500, allora saranno necessarie 2 iterazioni per completare 1 epoca.

```
def mini_batch_GD (X, y, batch_size = 32 epochs = 100):
    w = rand (X.shape[1], 1)
    b = rand (1, 1)
    batches = X.shape[0]/batch_size
    for _ in range (epochs):
        for batch in range (batches):
            if (batch == batches-1):
                X_batch = X[batch*batch_size:,:]
                y_batch = y[batch*batch_size:]
            else:
                X_batch = X[batch*batch_size:(batch+1)*batch_size,:]
                y_batch = y[batch*batch_size:(batch+1)*batch_size]
            z = dot (x_batch,w) + b
            a = activation (z)
            dw = X_batch.T.dot(a-y_batch)
            db = sum(a-y_batch)
            w = w - α*dw
            b = b - α*db
return w, b
```

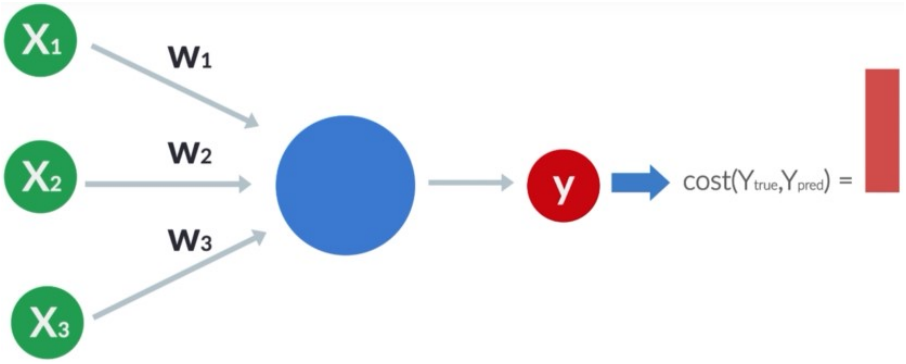
Il Gradient Descent, o un suo derivato, rappresenta il cuore di un processo chiamato *Propagazione*.

Forward & Back Propagation

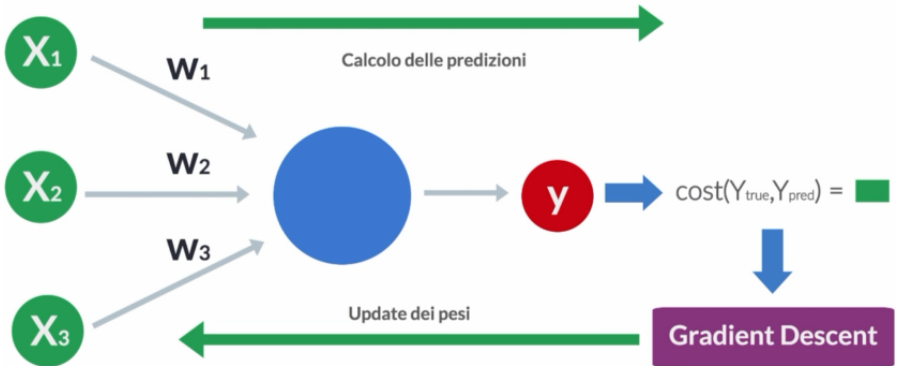
Si considera un modello di ML composto da un unico nodo:



Si esegue la prima epoca del Gradient Descent sul modello: vengono inizializzati i coefficienti w_i ad un valore casuale, si calcola l'output per ogni esempio del set di addestramento e si utilizza una funzione di costo per calcolare l'errore (generalmente molto alto).



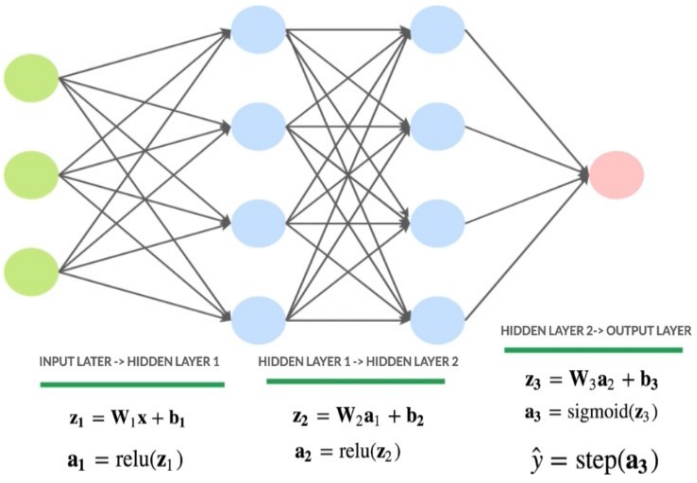
Successivamente, lo step del Gradient Descent permetterà di aggiornare l'entità dei coefficienti per ripetere le predizioni.



Nella seconda epoca si utilizzano i nuovi coefficienti per stimare la risposta, la quale fornirà una funzione di costo con un valore significativamente minore del precedente. Con un ulteriore step del GD si "torna indietro" e nuovamente si aggiornano i pesi al fine di avere, al termine di diverse epoche, una funzione di costo ottimizzata al suo valore minimo.

Questo processo che inizia dalla stima dei coefficienti e termina con la valutazione della funzione di costo è chiamato *forward propagation*, ovvero propagazione in avanti. Al contrario, quello che parte dalla stima dell'errore arrivando all'aggiornamento dei coefficienti è conosciuto come *backward propagation* o *backpropagation* (propagazione all'indietro). Quest'ultimo è l'algoritmo che più di tutti ha portato alla nascita del settore del Deep Learning; esso permette di sapere quanto ogni peso di ogni strato ha

contribuito all'errore del modello propagandolo dall'ultimo strato all'indietro. Mentre il Gradient Descent è prevalentemente utilizzato per un semplice modello di ML composto da un unico nodo (come la regressione logistica o lineare), quando si affronta una Rete Neurale Artificiale è consigliato l'uso di algoritmi più complessi:



Un semplice modello di ML è composto da un'unica funzione $f(x) = y$ che associa l'input x del modello all'output y. Una RNA, al contrario, è formata da più funzioni annidate, ognuna con i propri coefficienti che mappano uno strato al successivo.

Un esempio è rappresentato dal grafico in figura che modellizza un problema di classificazione binaria: in basso sono presenti tutte le funzioni necessarie per determinare l'output, ovvero il classico processo di *forward propagation*; si setta il peso W_1 e il bias b_1 a valori casuali e si calcola la risposta lineare z_1, la quale passerà attraverso la funzione di attivazione ReLU producendo l'output del primo strato a_1. Questo viene utilizzato come input per calcolare l'output lineare del secondo strato che, passandolo alla funzione ReLU, tirerà fuori a_2. Con lo stesso procedimento si determina la risposta lineare z_3 dell'ultimo strato nascosto, la quale sarà l'oggetto della funzione di attivazione sigmoidale che ne valuterà la probabilità di appartenenza alla classe positiva (anomala). Nel caso si volesse un output binario (0,1) rappresentante la classe di appartenenza, invece che la probabilità, si deve abbinare anche una funzione *step*. Come appena visto nelle RNA l'output di una funzione è sempre l'input

della successiva, dunque, si può facilmente raggruppare tutto il processo, quindi tutte le funzioni in gioco, in un'unica funzione composta perdendo tuttavia l'informazione sul singolo nodo.

Il problema nasce nel momento in cui si deve applicare il GD in quanto per eseguirlo è condizione necessaria la conoscenza di quanto ogni singolo coefficiente ha contribuito all'errore finale calcolato nello strato di output. La *back propagation* risolve tale problema calcolando tutte le derivate parziali (quindi il gradiente) di ogni coefficiente rispetto alla funzione di costo attraverso l'applicazione del metodo *chain rule*; si suppone di dover derivare la funzione annidata *f* rispetto alla variabile *x*:

$$f(x) = f(g(x))$$

per fare ciò si deve calcolare prima la derivata di *f* rispetto alla funzione *g* e successivamente moltiplicarla per la derivata di *g* rispetto a *x*:

$$\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}$$

La *chain rule* è stata utilizzata in precedenza (nel capitolo del GD) per il calcolo della derivata della funzione di costo rispetto ai coefficienti:

$$\boldsymbol{J(w,b) = J(\phi(wX + b))}$$

questa è composta da due funzioni annidate: la prima *wX+b* per calcolare la risposta lineare del modello e la seconda, rappresentata dalla funzione di attivazione ϕ, fornirà l'output effettivo che sarà confrontato con l'output reale del modello al fine di determinare l'errore commesso. Chiamando *z* la funzione lineare, si può applicare la *chain rule*:

$$\boldsymbol{J(w,b) = J(\phi(z(w,b)))}$$

$$\boldsymbol{\frac{dJ}{dw} = \frac{dJ}{d\phi}\frac{d\phi}{dz}\frac{dz}{dw}}$$

$$\frac{dJ}{db} = \frac{dJ}{d\phi}\frac{d\phi}{dz}\frac{dz}{db}$$

Quindi, il funzionamento della *back propagation* consiste nel calcolare le derivate della funzione di costo rispetto ad ogni singolo coefficiente di ogni strato prima rispetto unicamente all'ultimo strato, misurare l'errore finale commesso e propagarlo all'indietro utilizzando la chain rule.

Scomparsa del Gradiente

La scomparsa del gradiente (Vanishing Gradient problem) è un tipico problema delle RNA troppo profonde; esso provoca una forte riduzione del gradiente e conseguentemente un aggiornamento minimo dei parametri rendendo la rete difficile da addestrare. Si osservi il seguente caso limite:

La figura rappresenta una RNA profonda a 6 strati nascosti, ognuno dei quali ha un solo nodo, e un unico nodo di input e di output. Semplificando ancora di più il problema, si ipotizza di utilizzare la funzione identità *f(X)=X* e un bias nullo (*b=0*) per tutti gli strati. Inoltre, si assume uguale a 1 l'input X e ogni peso w ad un valore iniziale di 0,5; si procede con la *forward propagation*:

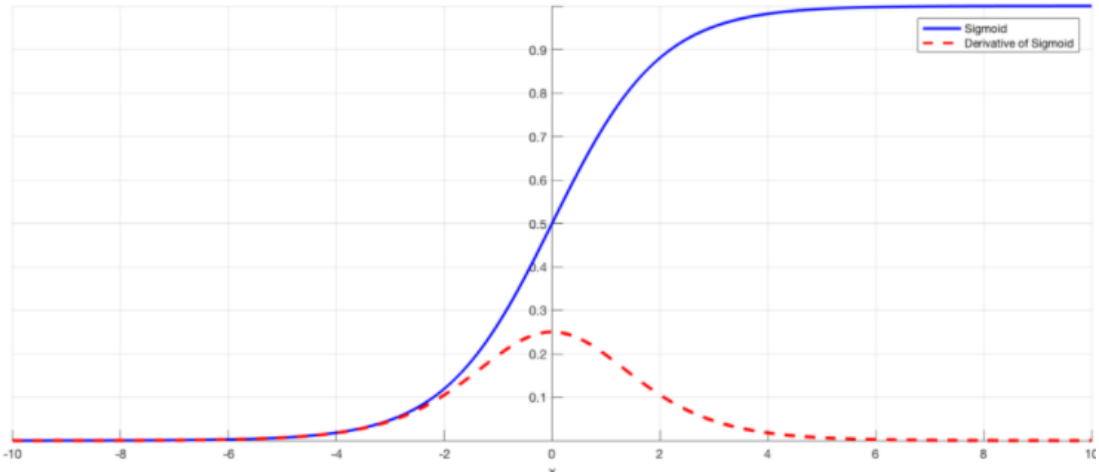
dato che come funzione di attivazione viene utilizzata la lineare, la risposta sarà inevitabilmente il prodotto tra l'input X e tutti i coefficienti w_i:

$$X \cdot w_1 \cdot w_2 \cdot w_3 \cdot w_4 \cdot w_5 \cdot w_6 \cdot w_7 = y$$

$$1 \cdot 0{,}5 \cdot 0{,}5 \cdot 0{,}5 \cdot 0{,}5 \cdot 0{,}5 \cdot 0{,}5 \cdot 0{,}5 = 0{,}5^7 \approx 0{,}0078$$

La risposta finale della rete è molto piccola così come l'output a_i che scaturisce ad ogni strato:

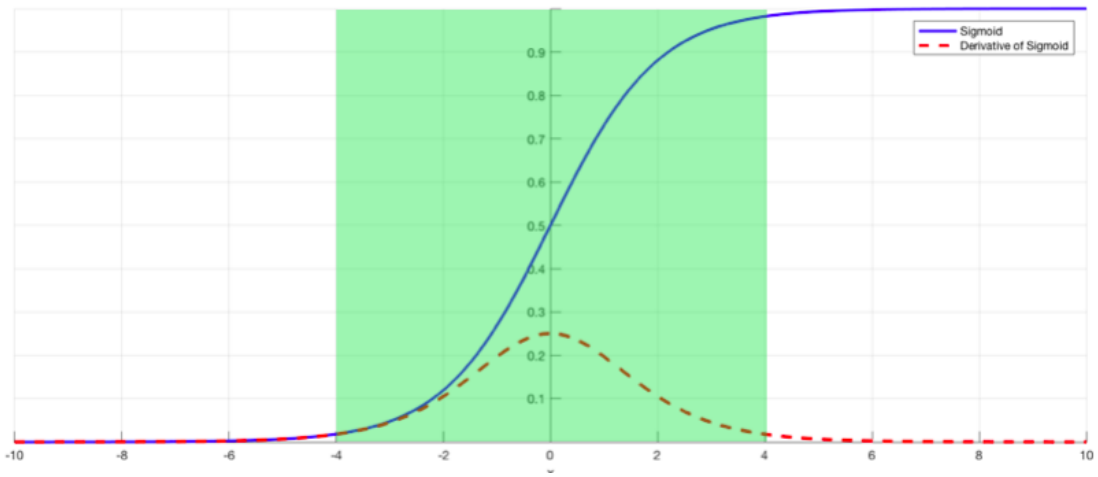
L'output del primo strato nascosto è pari all'input X moltiplicato il peso 0,5, ovvero ancora 0,5 = 1/2; quello del secondo sarà l'output a_1 del primo strato per il peso 0,5, quindi 1/4. Con l'aumentare del numero degli strati l'output è sempre più piccolo, arrivando fino a 1/64 nell'ultimo strato nascosto. Avere un gradiente di entità eccessivamente ridotta scaturisce dei problemi durante la fase di back propagation; questa viene utilizzata al fine di trovare i gradienti della rete spostandosi progressivamente dallo strato finale a quello iniziale. Come visto in precedenza, per eseguire la back propagation è necessario applicare la chain rule al fine di propagare all'indietro l'errore e ottenere il gradiente della funzione di costo rispetto ai vari parametri. Tuttavia, avere un rilevante numero di strati nascosti significa moltiplicare ripetutamente piccole derivate e rischiare che il gradiente finisca per scomparire (effetto *vanishing*) tendendo verso lo zero man mano che si avvicina allo strato iniziale; il risultato sarà che ogni step del GD risulterà anch'esso molto piccolo e richiederà moltissimo tempo per riuscire a trovare un punto di minimo. La situazione si aggrava pesantemente nel momento in cui si utilizza come funzione di attivazione per uno degli strati nascosti una che ritorna valori tra 0 e 1; infatti, utilizzando attivatori come la sigmoide o la tangente iperbolica, a un grande cambiamento dell'input corrisponde un piccolo cambiamento nell'output in quanto la derivata dell'attivatore diventa sempre più piccola.



L'immagine rappresenta funzione sigmoidale e la sua derivata; quando gli input della funzione diventano più grandi o più piccoli, ovvero quando |x| diventa più grande, la derivata diventa prossima a zero.

La soluzione più semplice è quella di utilizzare altre funzioni di attivazione per gli strati nascosti, come la ReLU, che non provoca una piccola derivata avendo un range tra 0 e +∞.

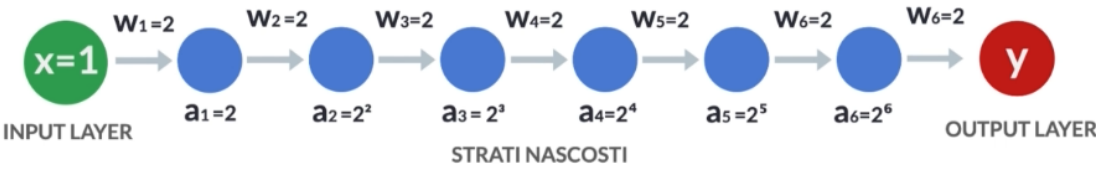
Un'altra è rappresentata dall'impiego dei *livelli di normalizzazione batch*: dato che il problema sorge quando un grande spazio di input viene mappato su uno piccolo (|x| elevato), la normalizzazione batch normalizza l'input in modo tale che |x| non raggiunga i bordi esterni della funzione di attivazione.



L'input viene regolarizzato per fare in modo che la maggior parte di esso ricada nella regione verde, dove la derivata non è troppo piccola.

Esplosione del Gradiente

L'esplosione del gradiente (Exploding Gradient problem) è esattamente il problema opposto al vanishing, per cui si ha quando la RNA è troppo profonda e il gradiente rischia di diventare troppo grande a cui seguirà un aggiornamento dei coefficienti e bias di entità troppo elevata. Ripetendo l'esempio precedente, si ipotizza un input X pari a 1 e pesi w = 2:



$$1 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 2^7 = 128$$

La forward propagation produce un output nello strato finale di 128, quindi molto superiore rispetto all'input; inoltre, anche le risposte dei singoli strati crescono esponenzialmente. Durante la backward propagation si dovrebbero utilizzare questi valori intermedi per calcolare il gradiente attraverso la chain rule ma moltiplicando valori sempre più grandi il gradiente "esploderà" diventando eccessivamente grande.

Risulta difficile trovare una soluzione definitiva; quella più ricorrente consiste nell'*inizializzazione intelligente dei parametri*. Invece che farlo a valori casuali, si utilizza la seguente formula:

$$\boldsymbol{w^{[l]} = rand(-1, 1)\sqrt{\frac{2}{n^{[l-1]}}}}$$

dove *rand(-1,1)* seleziona un numero casuale compreso tra -1 e 1, n è il numero di nodi nello strato corrente e l è il numero dello strato corrente. Questa formula permette anche di mantenere la varianza costante mentre si calcola l'output della rete.

Con una ricerca del 2010, Xavier Glorot e Yoshua Bengio hanno proposto altri metodi per l'inizializzazione intelligente dei parametri, come:

$$\boldsymbol{w^{[l]} = rand(-1, 1)\sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}}$$

$$\boldsymbol{w^{[l]} = rand\left[-\sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}, \sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}\right]}$$

La prima, conosciuta come Glorot-Xavier Normal Initializer, è molto simile a quella base con l'unica aggiunta nel denominatore (del fattore che permette di mantenere costante la varianza) di un addendo rappresentante il numero di nodi nello strato corrente elevato al numero dello strato stesso. La seconda formula è stata chiamata Glorot-Xavier Uniform Initializer; in essa il valore dei coefficienti è estratto da una distribuzione uniforme con limiti $\left[-\sqrt{\frac{2}{n^{[l-1]}+n^{[l]}}}, \sqrt{\frac{2}{n^{[l-1]}+n^{[l]}}}\right]$.

Momentum

Il *momentum* è una tecnica per accelerare la convergenza del GD verso un punto di minimo globale. L'idea deriva dalla fisica e in particolare è la misura di una massa in movimento, ovvero la quantità di moto ($momentum = massa \cdot velocità$).

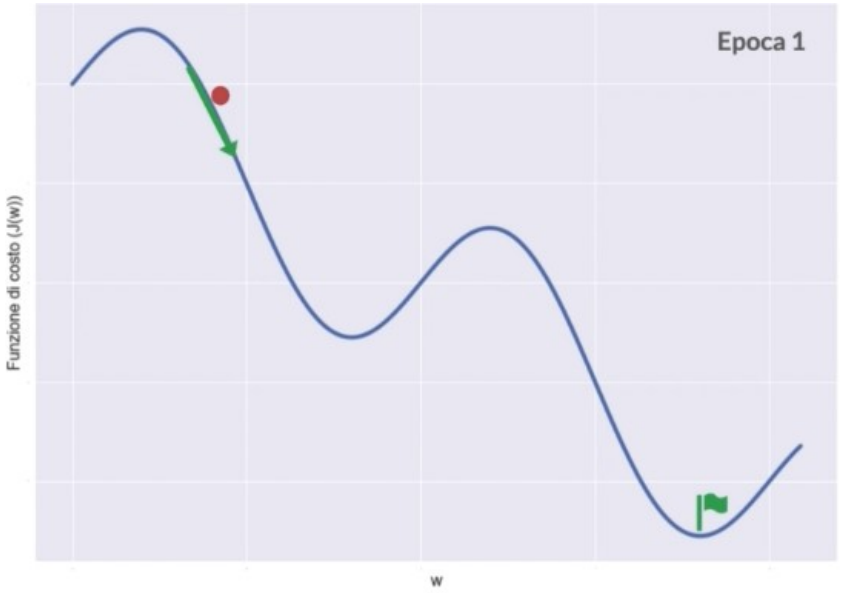
Tale concetto si può integrare all'interno del GD:

$$\boldsymbol{V_t = \gamma V_{t-1} + \eta \nabla J(\theta)}$$

$$\boldsymbol{\theta = \theta - V_t}$$

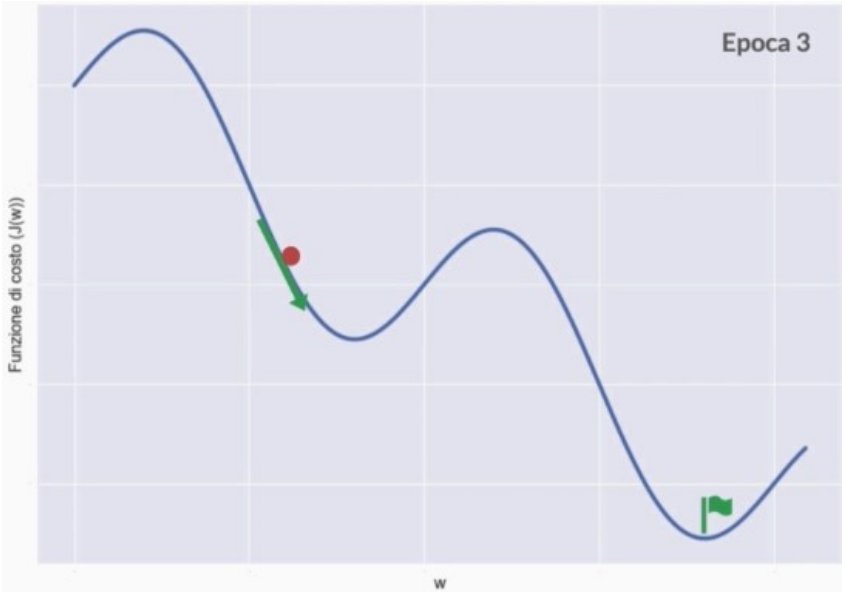
Si prende la somma tra quello che sarebbe stato l'update del Gradient Descent, quindi il gradiente della funzione di costo J rispetto ai coefficienti θ moltiplicato per il learning rate η, e una frazione γ (generalmente uguale a 0,9) dell'update allo step precedente; θ è un vettore che contiene sia pesi che bias. Il valore V_t risultante viene utilizzato per eseguire l'aggiornamento effettivo dei coefficienti.

Di seguito si effettua una simulazione per vedere cosa succede quando si utilizza il momentum nel GD:



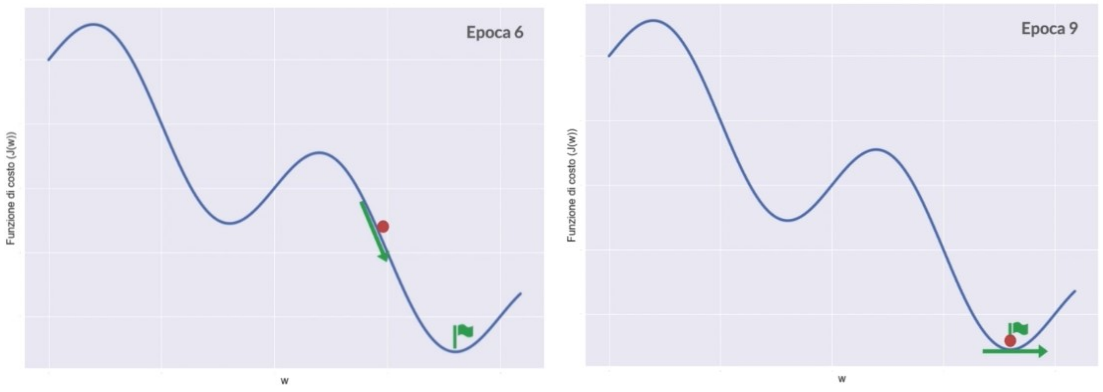

```
w = rand()
v = 0
for epoch in epochs:
    dw = cost_derivate(w)
    v = gamma*v + alpha*dw
    w = w - v
```

Al fine di visualizzarla in due dimensioni si pone la funzione di costo J(w) su un asse e un unico coefficiente w sull'altro. Lo scopo è trovare il coefficiente che permette di minimizzare la funzione di costo; in questo caso, però, vi è un punto di minimo locale in cui il GD potrebbe facilmente bloccarsi. Inizialmente, si pone w ad un valore casuale e il momentum v a 0; per ogni epoca si calcola la derivata della funzione di costo rispetto al coefficiente e per eseguire l'update si utilizzano le formule del momentum.



Man mano che il GD prosegue in una stessa direzione, il valore del momentum crescerà permettendogli, in questo modo, di "acquisire velocità".

Nel momento in cui il GD oltrepassa un punto di minimo, se non si fosse utilizzato il momentum, in tale punto la direzione dell'update sarebbe dipesa solamente dalla derivata e quindi avrebbe cambiato di segno facendo erroneamente tornare il GD verso il punto di minimo locale. Al contrario, il momentum, inizialmente posto a zero, è cresciuto durante la prima rapida discesa del GD a tal punto che riesce a ribaltare la direzione dell'update e a far proseguire il GD nella direzione iniziale.



Il GD continua la sua discesa fino al raggiungimento del punto di minimo globale.

Esiste anche una variante che spesso nei casi pratici permette di ottenere risultati migliori: il *Nesterov Momentum*.

$$\boldsymbol{V_t = \gamma V_{t-1} + \eta \nabla J(\theta - \gamma V_{t-1})}$$

$$\boldsymbol{\theta = \theta - V_t}$$

Il funzionamento è il solito del momentum con l'unica differenza che il gradiente, invece di essere calcolato rispetto ai coefficienti attuali, è valutato rispetto ad una stima dei coefficienti futuri basata sull'ipotesi che l'update corrente non sarà molto diverso da quello eseguito allo step precedente. L'implementazione in python risulta essere molto simile a quella base:

```
w = rand()
v = 0
for epoch in epochs:
    dw = cost_derivate(w-v)
    v = gamma*v + alpha*dw
    w = w - v
```

LR adattivo: Algoritmi di Ottimizzazione

Il motivo di adoperare un *learning rate (LR) adattivo* risiede nel fatto che il learning rate statico è uno degli iperparametri più complicati da selezionare; inoltre, lo statico sarà uguale per ogni features e questo può essere uno svantaggio quando si trattano features molto rare come, ad esempio, variabili categoriche che appaiono molto raramente.

Una soluzione al primo problema è rappresentata dalla possibilità implementare un learning rate che si riduce ad ogni iterazione del GD (*learning rate decay*):

$$\boldsymbol{\alpha} = \frac{\boldsymbol{\alpha}}{\mathbf{1} + \boldsymbol{kt}}$$

Dove α è il learning rate, k il tasso di decadimento (decay rate) e t il numero dell'iterazione corrente; k indica quanto velocemente si deve far ridurre il learning rate ma, essendo anch'esso un iperparametro, risolve il primo ma non il secondo problema.

Una soluzione più completa è l'*Adagrad*: una variante del GD che ci permette di ottenere un learning rate dinamico che si adatta ad ogni features.

$$\boldsymbol{g_t} = \boldsymbol{\nabla J(\theta)}$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \frac{\boldsymbol{\eta}}{\sqrt{\boldsymbol{G_t} + \boldsymbol{\varepsilon}}} \cdot \boldsymbol{g_t}$$

Si calcola il gradiente della funzione di costo rispetto ai coefficienti e si immagazzina il valore all'interno della variabile g_t, dove t rappresenta l'epoca corrente. Al fine di eseguire l'update, si divide il learning rate per il fattore sotto radice quadrata; ε è un valore estremamente piccolo che serve unicamente per evitare un'eventuale divisione per 0, mentre G_t è una matrice

diagonale che contiene la somma al quadrato di ogni derivata parziale fino all'epoca t:

$$G_t = \begin{bmatrix} \sum_{i=1}^{t} g_{i,1}^2 & 0 & 0 \\ 0 & \sum_{i=1}^{t} g_{i,2}^2 & 0 \\ 0 & 0 & \sum_{i=1}^{t} g_{i,3}^2 \end{bmatrix}$$ una matrice diagonale è una matrice in cui tutti i valori sono zero tranne quelli sulla diagonale principale. In questo caso abbiamo un dataset con 3 features.

Un G_t composto in questo modo crea un problema all'Adagrad in quanto, raccogliendo le derivate parziali a tutte le iterazioni, tende a crescere eccessivamente ed essendo al denominatore del learning rate rischia di ridurlo molto.

Le soluzioni maggiormente utilizzate sono l'RMSProp e l'Adadelta (entrambe espansioni dell'Adagrad). L'idea è quella di accumulare i gradienti non di tutte le iterazioni ma soltanto di un ultimo intervallo temporale mediante l'utilizzo di una media mobile esponenziale:

$$\boldsymbol{g_t = \nabla J(\theta)}$$

$$\boldsymbol{E[g^2] = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2}$$

$$\boldsymbol{\theta = \theta - \frac{\eta}{\sqrt{E[g^2] + \varepsilon}} \cdot g_t}$$

la matrice diagonale è stata sostituita con una media mobile esponenziale la quale assegna un peso maggiore all'ultimo gradiente rispetto a quelli precedenti.

L'Adagrad, RMSProp e Adadelta raramente vengono utilizzati in quanto l'algoritmo di ottimizzazione per eccellenza è l'*Adam*, ovvero una

combinazione tra l'RMSProp e il momentum. L'Adam è schematizzato con le seguenti formule:

$$\boldsymbol{g_t = \nabla J(\theta)}$$

$$\boldsymbol{m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t} \qquad \textbf{(Momentum)}$$

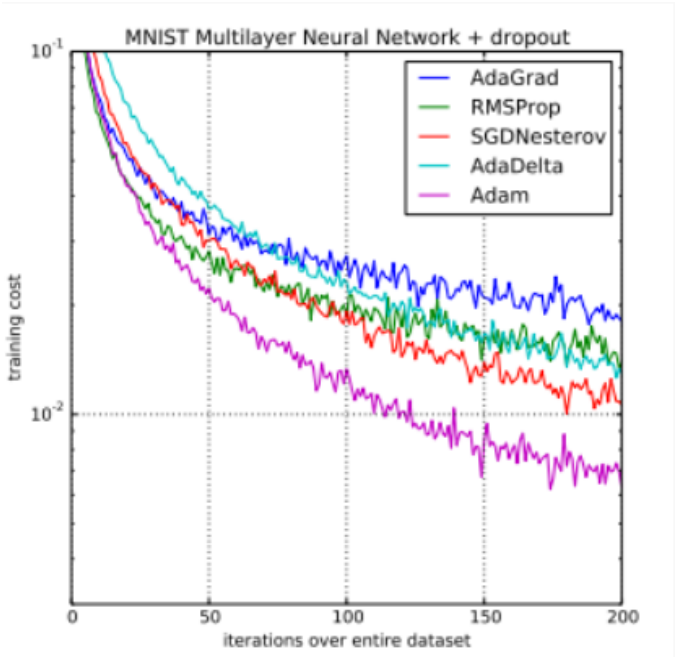
$$\boldsymbol{v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2} \qquad \textbf{(RMSProp)}$$

$$\boldsymbol{\hat{m}_t = \frac{m_t}{1 - \beta_1^t}}$$

$$\boldsymbol{\hat{v}_t = \frac{v_t}{1 - \beta_2^t}}$$

$$\boldsymbol{\theta = \theta - \frac{\eta}{\sqrt{\hat{v}_t + \varepsilon}} \cdot \hat{m}_t}$$

dove β_1 e β_2 sono rispettivamente il tasso di decadimento esponenziale per le stime del primo (0,9) e del secondo (0,999) momento. Attualmente, l'Adam è il metodo di ottimizzazione più performante:



L'*Adamax* è una delle estensioni dell'Adam maggiormente utilizzate in quanto risulta più robusta se le features del modello sono rappresentate da una matrice sparsa, quindi una matrice di grandi dimensioni contenente per la maggior parte degli zeri. L'update rule dell'Adamax è:

$$\hat{\boldsymbol{u}}_{\boldsymbol{t}} = \boldsymbol{max}\,(\boldsymbol{\beta_2 v_{t-1}}, |\boldsymbol{g_t}|)$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \frac{\boldsymbol{\eta}}{\hat{\boldsymbol{u}}_{\boldsymbol{t}}} \cdot \hat{\boldsymbol{m}}_{\boldsymbol{t}}$$

dove il learning rate η viene diviso per $\hat{u}_t$, il quale rappresenta il massimo tra il prodotto di β_2 e i gradienti accumulati e la norma di g_t.

Un'ulteriore variante dell'Adam è il Nadam che, invece, mette insieme il metodo RMSProp e il Nesterov Momentum. Pubblicato nel 2016, risulta essere il metodo più recente e utilizza la seguente update rule:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \frac{\boldsymbol{\eta}}{\sqrt{\hat{\boldsymbol{v}}_{\boldsymbol{t}} + \boldsymbol{\varepsilon}}} \cdot (\boldsymbol{\beta_1}\hat{\boldsymbol{m}}_{\boldsymbol{t}} + \frac{(\boldsymbol{1} - \boldsymbol{\beta_1})\boldsymbol{g_t}}{\boldsymbol{1} - \boldsymbol{\beta_1^t}})$$

METODI DI APPRENDIMENTO ENSEMBLE

In questa sezione si introduce un nuovo metodo di apprendimento: "*apprendimento di insieme*". L'Ensemble è un paradigma di apprendimento automatico in cui più modelli (chiamati "weak learners", "studenti" o "allievi") vengono addestrati per risolvere uno stesso problema e combinati tra loro al fine di ottenere risultati migliori. L'idea fondante è la seguente: un modello costruito mettendo insieme più modelli "deboli" (o di base) risulta essere maggiormente accurato e robusto dei modelli presi singolarmente; questo permette di ottenere una migliore generalizzazione su dati sconosciuti, riducendo il rischio di overfitting. Infatti, i singoli modelli di base spesso non riescono a offrire le migliori performance a causa di un bias elevato[5] oppure per un'alta varianza[6]. Per impostare un metodo di apprendimento ensemble è necessario selezionare inizialmente i *modelli base da aggregare*. Nella maggior parte dei casi viene utilizzato un unico algoritmo di apprendimento di base in modo tale da avere modelli deboli *omogenei*; dunque, il modello ensemble che ne deriva sarà omogeneo. Tuttavia, esistono anche altri metodi (come lo stacking) che, utilizzando diversi tipi di algoritmi di apprendimento di base, combinano modelli tra loro eterogenei creando un modello di insieme eterogeneo. Nelle prossime sezioni si introducono i più importanti algoritmi utilizzati per combinare modelli base *omogenei.*

[5] Bias elevato: modello eccessivamente semplice, ovvero con pochi gradi di libertà per riuscire a risolvere la complessità sottostante dei dati. Vi è il rischio di underfitting.
[6] Varianza elevata: modello eccessivamente complicato, ovvero con molti gradi di libertà che ne riducono la robustezza e stabilità. Vi è il rischio di overfitting.

Bagging (Bootstrap Aggregating)

Il punto di partenza è la selezione di una (e unica) tipologia di modello base che si vuole utilizzare (generalmente si tratta di un *albero decisionale*); si estraggono dal dataset originale N nuovi set di dati di addestramento mediante la tecnica del *campionamento casuale con rimpiazzo* (bootstrap). Campionando con bootstrap alcune osservazioni possono essere ripetute nei nuovi campioni di training; nel caso del Bagging, qualsiasi osservazione ha la stessa probabilità di apparire all'interno di un nuovo set di dati. Tali campioni risulteranno tra loro "*quasi indipendenti*". A questo punto, il metodo Bagging consiste nell'addestrare un insieme di modelli base omogenei tra loro "in modo parallelo", ovvero ogni modello viene addestrato da uno degli N

campioni di bootstrap quasi indipendenti, producendo in questo modo N modelli deboli (tutti alberi decisionali) indipendenti (o quasi); Quindi, l'addestramento avviene *parallelamente*:

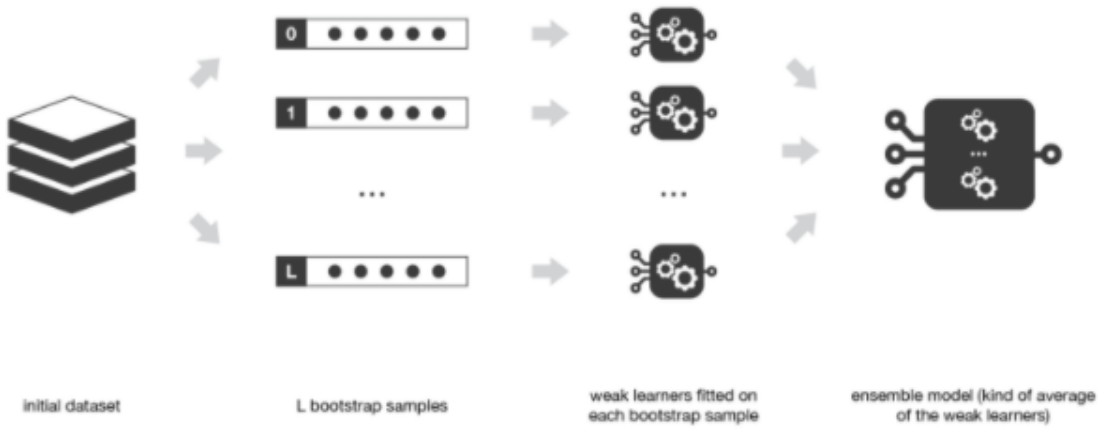


Figura 1: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

Gli N studenti vengono tutti mantenuti in quanto il Bagging segue una procedura *train&keep*. Ogni singolo albero effettua una previsione (voto) per ogni osservazione del set di test, in modo indipendente l'uno dall'altro. La

previsione finale (ad esempio la classe delle società sane o anomale) per ogni istanza (esempio) è la classe che riceve più della metà dei voti (*Majority vote*) contati su ogni modello[7]. Se nessuna delle classi ottiene più della metà dei voti per una determinata osservazione, allora il metodo dell'insieme non è stato in grado di effettuarne una previsione stabile. Un'altra tecnica che può essere utilizzata è il "voto di pluralità" con la quale la predizione finale sarà la classe più votata, indipendentemente dal fatto che abbia o meno raggiunto la metà dei voti.

Random Forest

Il *Random Forest* (Foreste Casuali) è una variante del Bagging e, come tale, ne condivide la stessa struttura ma, al fine di rendere i vari alberi meno correlati tra loro, invece di campionare solo sulle osservazioni, si effettua un campionamento anche sulle features, selezionate casualmente per ogni albero[8]. Campionando anche sulle variabili, gli alberi non guarderanno le stesse informazioni per prendere le decisioni, riducendo conseguentemente la correlazione tra i diversi output.

[7] Ad esempio, supponendo di avere 10 modelli deboli, se l'osservazione x risulta classificata come "sana" in 6 o più modelli base (dei 10 complessivi), allora nel modello ensemble l'osservazione sarà classificata come "sana".

[8] Lo split ottimale dell'albero è basato, quindi, su una selezione casuale di un subset di features.

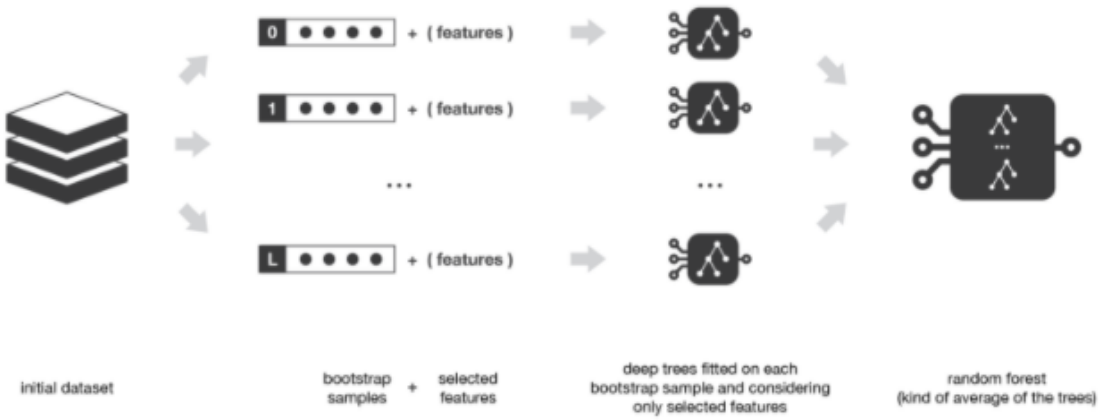


Figura 2: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

Boosting

Come i precedenti, anche questo metodo si basa sull'individuazione di un unico tipo di algoritmo di apprendimento (generalmente gli alberi decisionali) da utilizzare; tuttavia, il Boosting mira a creare un modello ensemble addestrando ogni modello debole con *lo stesso dataset*, nel quale però i pesi delle singole osservazioni vengono regolati in base agli ultimi errori di predizione commessi. Nel dettaglio, gli N modelli base vengono costruiti *iterativamente* e *sequenzialmente*: inizialmente si ha un dataset di training composto da osservazioni aventi tutte lo stesso peso[9], si *crea* uno studente base (uno alla volta), si *addestra* sui dati di training, se ne *calcola* il tasso di errore ponderato e, qualora superi delle condizioni descritte in seguito, si *aggrega* al modello ensemble; successivamente, si *aggiornano i pesi dei singoli esempi* del training set in modo tale da evidenziare allo studente successivo (che verrà

[9] Se il training set è composto da 100 osservazioni, allora ognuna di esse avrà un peso di 0.01 (1/100).

addestrato su tale training set) i successi e fallimenti del modello di insieme corrente:

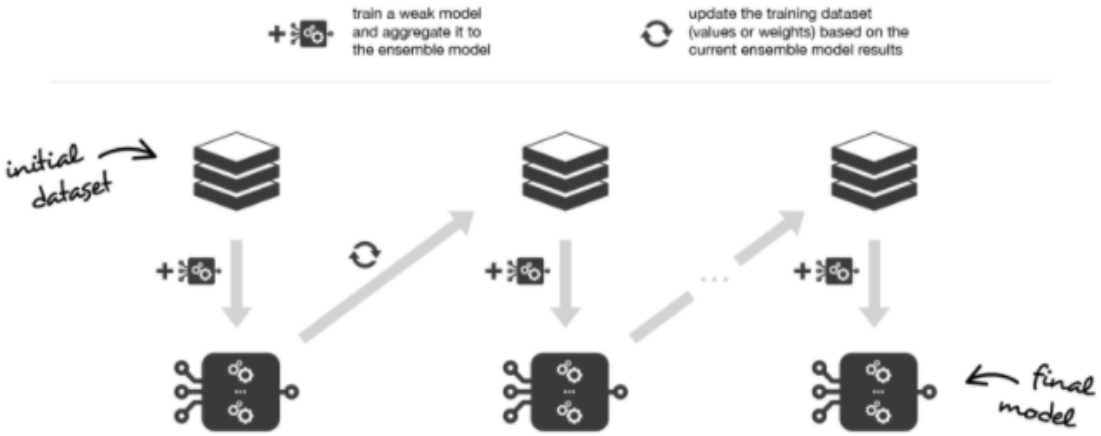


Figura 3: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

In particolare, i pesi delle singole osservazioni classificate erroneamente verranno aumentati per enfatizzare i casi più difficili e, al contrario, i pesi di quelle correttamente predette saranno diminuiti. L'idea quindi è quella di indurre ogni modello debole successivo a concentrarsi maggiormente sugli errori commessi dal modello ensemble corrente (costituito dai suoi predecessori), attraverso un addestramento sui *nuovi dati ponderati*; in questo modo, gli N modelli deboli registreranno tra loro una bassa correlazione e, di conseguenza, maggiore sarà la robustezza del modello di insieme definitivo (*Strong Learner*) ottenuto al termine di tutte le iterazioni. In un problema di classificazione, l'*aggregazione iterativa* dello studente al modello di insieme avviene mediante una votazione pesata[10] (weighted voting) in quanto il Boosting, oltre ad aggiornare i pesi delle singole osservazioni, assegna anche una *seconda serie di pesi* direttamente ai *modelli base* (che man mano compongono il modello ensemble). In particolare, durante l'addestramento

[10] Il funzionamento è identico al voto di maggioranza (utilizzato nei metodi di Bagging) con la sola aggiunta che pesando i vari studenti, le loro predizioni avranno una diversa rilevanza in sede di predizione finale del modello di insieme.

l'algoritmo assegna a un modello con buone predizioni sui dati di training un peso maggiore rispetto a uno con pessime prestazioni[11]; maggiore peso ha un modello base, maggiore sarà la sua influenza sulle predizioni del modello ensemble finale. Dato che il Boosting, a differenza del Bagging, segue una logica *train&evaluate*, in alcuni casi un modello debole, per essere aggregato al modello ensemble, deve soddisfare alcune condizioni; infatti, esistono alcune tecniche di boosting che includono una condizione extra per mantenere o scartare un singolo studente. Ad esempio, si può impostare una soglia massima[12] di errore sopra la quale lo studente viene scartato e l'iterazione ripetuta fino quando non si costruisce uno studente con un errore inferiore a tale soglia; al contrario, se il modello debole ha un errore accettabile, viene mantenuto, calcolato il suo peso e aggiornato i pesi delle singole osservazioni del set di addestramento. Al termine della sequenza, le predizioni del modello definitivo (*Strong Learner*) saranno, come detto, una votazione pesata di quelle dei singoli modelli base che lo compongono.

Tornando agli aspetti generali, sia il Bagging che il Boosting risultano essere idonei alla riduzione della varianza in quanto entrambi combinano diverse stime di modelli diversi (ma di uguale tipologia), producendo un modello forte maggiormente stabile. Tuttavia, il Boosting è principalmente incentrato sulla riduzione del bias, quindi i modelli deboli che vengono generalmente utilizzati con tale metodo sono quelli che presentano bassa varianza e alto bias.

[11] Al fine di valutare un nuovo studente, il Boosting deve necessariamente tenere traccia degli errori commessi dai suoi predecessori.

[12] Nell'algoritmo AdaBoost (spiegato in seguito) la soglia massima di errore è del 50%; quindi, se si crea uno studente base che ha meno del 50% di errore allora si mantiene, altrimenti si scarta e se ne crea un altro che prenderà il suo posto.

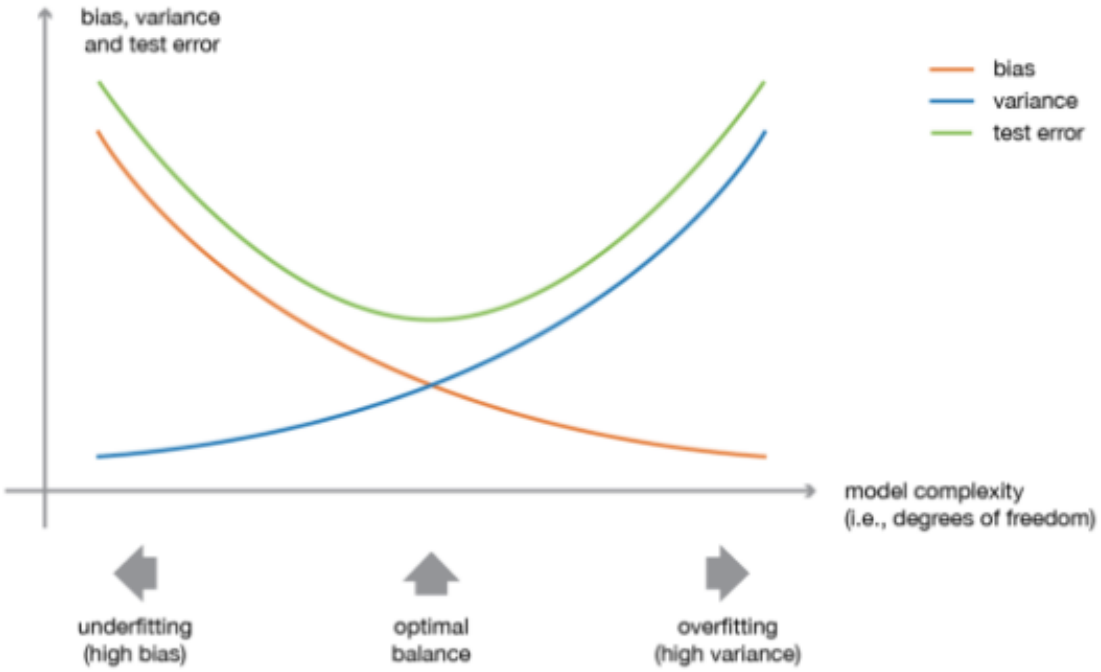


Figura 4: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

Ad esempio, se si utilizza gli alberi decisionali come modelli base, nella maggior parte dei casi si dovrà scegliere alberi a bassa profondità; infatti, minore è la profondità di un albero, minore sarà la sua complessità e quindi la struttura risulta conforme alla descrizione "bassa varianza e alto bias", ovvero modelli eccessivamente semplicistici che hanno un problema di underfitting. Inoltre, dato che l'addestramento non può essere parallelizzato, la scelta di tali strutture come modelli deboli è dovuta anche al fatto che sono molto meno costosi da adattare sequenzialmente dal punto di vista computazionale in quanto presentano pochi gradi di libertà. Al contrario, gli alberi profondi, avendo un'alta varianza e basso bias, sono strutture più complesse (tendenti all'overfitting) che risultano essere particolarmente utilizzate con i metodi di bagging, i quali mirano a ridurre la varianza.

Il primo algoritmo di Boosting ad essere stato sviluppato per la classificazione binaria fu l'*AdaBoost*: gli studenti deboli applicati in questo metodo sono chiamati "monconi o ceppi decisionali", ovvero alberi a un livello (con

un'unica suddivisione); questi modelli base sono estremamente semplicistici ma, fintanto che le loro prestazioni sono maggiori al 50% (quindi maggiori all'evento casuale), si mantengono per costruire un modello forte. Il processo è esattamente lo stesso di quello descritto nel Boosting:

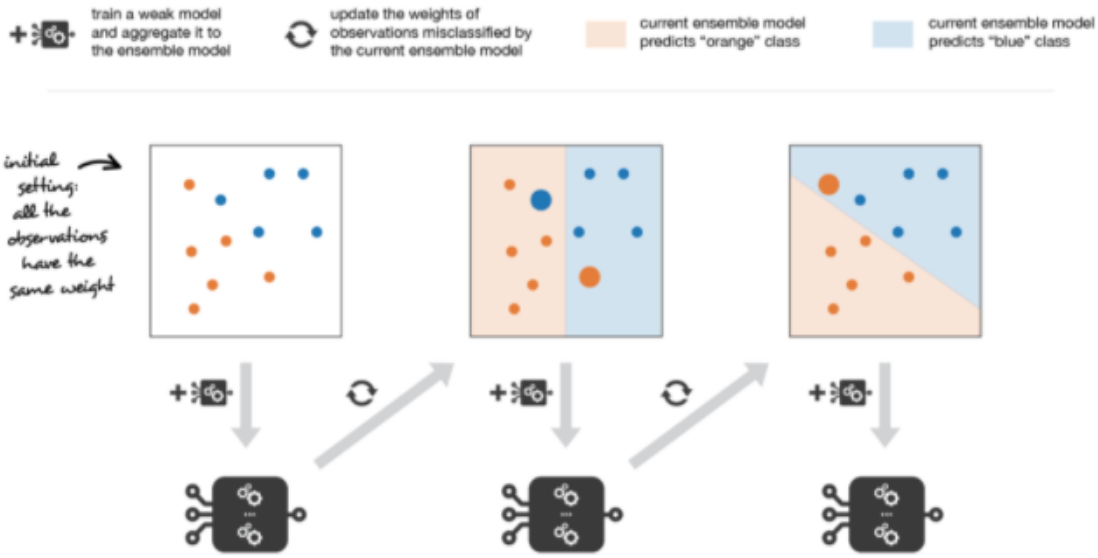


Figura 5: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

Come si vede dalla figura: si crea e addestra l'albero decisionale (moncone decisionale) su un train set avente le osservazioni equamente pesate, si aggrega (se supera le condizioni) al modello ensemble e, successivamente, si aggiornano i pesi delle singole osservazioni (aumentano per quelle *classificate erroneamente* e diminuiscono per quelle *classificate correttamente*) e dei modelli base (quelli con prestazioni migliori hanno peso maggiore nel modello di insieme finale). A seconda delle predizioni ottenute ad ogni iterazione, l'AdaBoost cercherà di risolvere un problema di ottimizzazione "locale", andando a trovare il miglior studente debole da aggiungere al modello ensemble al fine di ridurne l'errore di classificazione.

Un altro algoritmo fondato sul Boosting è il *Gradient Boosting*: l'idea generale rimane quella dei metodi Boosting, ovvero di creare un modello di insieme

forte attraverso l'aggregazione iterativa di studenti base (generalmente alberi decisionali), addestrati in modo tale da concentrarsi sulle predizioni errate; tuttavia, metodi diversi utilizzano modalità diverse. In particolare, il Gradient Boosting ridefinisce il Boosting come un *problema di ottimizzazione* nel quale l'obiettivo è minimizzare una certa funzione di costo[13] differenziabile mediante l'aggiunta iterativa di studenti deboli (uno alla volta) durante la fase di addestramento, utilizzando una procedura simile a quella del Gradient Descent. Inizialmente, si crea un primo studente per prevedere le osservazioni presenti nel training set e si calcola la sua rispettiva funzione di costo, ovvero un particolare scostamento tra gli output predetti dal primo allievo e le risposte reali; successivamente, alla seconda iterazione, si costruisce un secondo studente che viene addestrato sull'errore residuo (calcolato per ogni osservazione) prodotto dal primo studente al fine di prevedere la perdita dopo il primo step di addestramento e di correggerne gli errori commessi. In dettaglio, gli errori residui (chiamati anche "*pseudo-residui*") sono calcolati come l'*opposto del gradiente dell'errore di adattamento rispetto alle predizioni del modello di insieme corrente*; tale errore di adattamento è esattamente la funzione di costo da minimizzare. Dato che gli pseudo-residui indicano in quale direzione correggere le previsioni del modello di insieme corrente (in modo tale da ridurne l'errore di classificazione), addestrare il secondo allievo debole su questi valori equivale a effettuare un addestramento basato sulla minimizzazione del gradiente della funzione di costo[14]; Il Gradient Boosting richiama nel nome il Gradient Descent per questa analogia, in quanto

[13] Funzione di costo: dipende dalla tipologia di problema che si sta affrontando, regressione o classificazione. In caso di classificazione generalmente viene utilizzata la "Binary Cross-Entropy".

[14] L'addestramento è basato sulla riduzione al minimo di una funzione di costo (utilizzando il Gradient Descent il cui gradiente si riferisce alla pendenza di questa funzione, ovvero alla quantità di errore (pseudo-residui). Un piccolo gradiente implica un piccolo errore e, di conseguenza, una piccola modifica al modello per correggere tale errore e viceversa.

quest'ultimo è un algoritmo iterativo di ottimizzazione del primo ordine utilizzato per trovare un minimo locale di una funzione differenziabile[15]; dunque, l'*adattamento dei modelli deboli agli pseudo-residui rimanenti dal modello di insieme corrente* è un altro mezzo per enfatizzare le osservazioni classificate erroneamente, profondamente diverso dall'algoritmo AdaBoost, il quale modifica la distribuzione del training set alterando i pesi delle singole osservazioni. Questa particolarità è il fulcro del Gradient Boosting e consente ai nuovi allievi base, iterativamente aggiunti ad ogni step di addestramento, di compensare i punti deboli dei loro predecessori. Un'altra differenza con L'AdaBoost risiede nella tipologia di modello debole che viene utilizzata per minimizzare la funzione di costo: mentre l'AdaBoost fa uso esclusivamente di alberi a un livello, nel Gradient Boosting è possibile utilizzare alberi più profondi in quanto i modelli base sono degli alberi decisionali costruiti individuando i "*migliori punti di divisione*[16]" in base al coefficiente di purezza (Gini o Entropia); nonostante ciò, è comune vincolare i modelli deboli all'avere un massimo numero di livelli, nodi, divisioni o nodi foglia per agevolare la fase di addestramento.

[15] Nel Gradient Boosting, appunto, la funzione differenziabile è l'errore di adattamento.

[16] Un qualsiasi nodo di un albero (partendo da quello iniziale, chiamato radice) viene splittato selezionando la variabile che massimizza il guadagno di informazione derivante dallo split stesso; tale guadagno risulta nullo quando un'osservazione, cadendo in uno qualsiasi dei rami dello split (2 o più), ha la stessa probabilità di appartenere ad una classe o ad un'altra. La metrica che misura quanto buono è il guadagno di informazione si chiama *impurità* e vale 0 quando un nodo contiene esempi di un'unica classe; in tal caso il nodo è definito *puro* (l'albero avrà raggiunto una *foglia*, uno dei nodi finali). L'obiettivo dell'addestramento è raggiungere le foglie effettuando il minor numero di split.

Schematicamente:

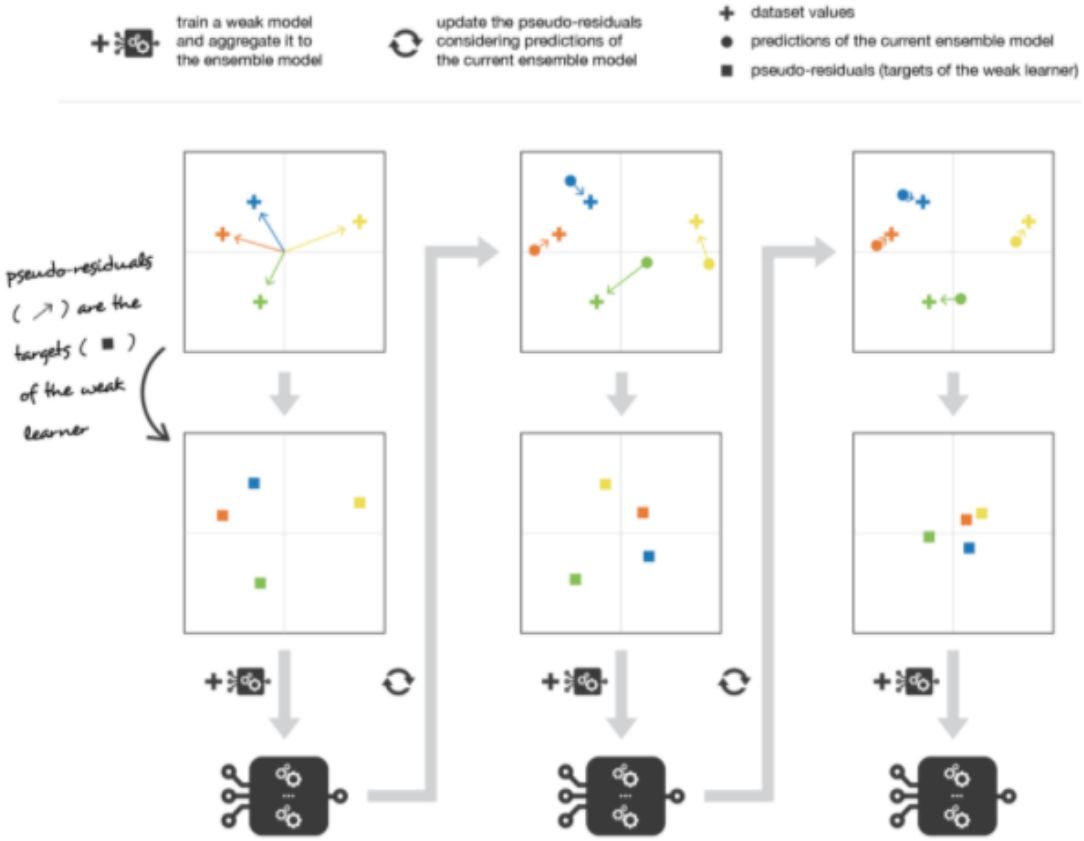


Figura 6: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

Inizialmente, gli pseudo-residui vengono posti uguali ai valori delle osservazioni del training set. Quindi, si ripetono per un determinato numero di volte (pari al numero di modelli base che si è deciso di creare nella sequenza) i seguenti passaggi: si adatta il nuovo studente debole agli pseudo-residui del modello di insieme corrente[17]; si calcola il valore ottimo (tramite un processo di ottimizzazione) dell'entità del "passo del Gradient Descent", ovvero di quanto il modello di insieme corrente deve aggiornarsi nella direzione del nuovo studente debole; si aggiorna il modello di insieme corrente aggregando lo studente debole moltiplicato per la dimensione del

[17] In termini pratici, "adattare agli pseudo-residui" significa addestrare un modello debole su un training set con una variabile target impostata ai valori degli pseudo-residui.

passo (effettuato un passo del Gradient Descent); si calcola i nuovi pseudo-residui di questo nuovo modello di insieme che indicano, per ogni osservazione, in quale direzione si vorrebbero aggiornare le previsioni. Successivamente, si aggiunge un nuovo allievo e lo si adatta ai nuovi errori residuali, ripetendo i vari passi fino all'ultima iterazione. Il contributo di ogni modello base alla previsione finale di quello di insieme è basato su tale processo di ottimizzazione che mira a minimizzare l'errore complessivo del modello ensemble finale. Come precedentemente descritto, mentre l'AdaBoost mira a risolvere iterativamente un problema di ottimizzazione locale, al contrario il Gradient Boosting, utilizzando un approccio di Gradient Descent, può essere più facilmente adattato a di–verse funzioni di perdita. Pertanto, il Gradient Boosting può essere considerato come una generalizzazione dell'AdaBoost a funzioni di perdita differenziabili arbitrarie.

Infine, dopo una panoramica dei principali metodi ensemble, si espone l'algoritmo che verrà utilizzato nel caso pratico di questo lavoro di tesi: l'*XGBoost* (Extreme Gradient Boosting); negli ultimi anni, questa tecnica sta dominando il campo del Machine Learning a tal punto che il suo impiego nelle principali piattaforme per competizioni di data science (ad esempio le competizioni Kaggle) risulta costantemente in aumento. Come si evince dal nome, l'XGBoost è un algoritmo ensemble di apprendimento automatico basato sugli alberi decisionali, il quale utilizza il framework del Gradient Boosting (è una sua implementazione avanzata) ma con particolari accortezze e approssimazioni:

- Mentre il Gradient Boosting fa uso del gradiente della funzione di costo (derivate parziali prime) del modello base per ridurre al minimo l'errore complessivo del modello ensemble corrente, l'XGBoost utilizza

il *gradiente del secondo ordine*, quindi calcola le derivate parziali seconde come approssimazione[18].

- Al contrario del Gradient Boosting, l'XGBoost penalizza i modelli più complessi attraverso le *regolarizzazioni L1 (LASSO) e L2 (Ridge)*, ovvero le funzioni di costo saranno regolarizzate al fine di prevenire l'overfitting e migliorare le prestazioni complessive.
- *Parallelizza la costruzione sequenziale di ogni albero* ad un livello molto basso dato che ogni ramo indipendente dell'albero viene addestrato separatamente; questo velocizza molto la formazione di ogni modello base.
- Trovare il miglior split per ogni nodo richiede che l'algoritmo esamini tutti i dati per ogni features, ottenendo una complessità computazionale dell'ordine di O (n_osservazioni ; n_features); tuttavia, spesso, piccoli cambiamenti nella divisione non alterano radicalmente le prestazioni dell'albero. L'XGBoost è un metodo basato sull'istogramma e come tale sfrutta questo fatto *raggruppando le features in un insieme di raccoglitori* (bins) ed *eseguendo lo split su quest'ultimi* invece che su tutte le caratteristiche; tale processo equivale a un sottocampionamento del numero di split valutate dal modello. Dato che le features possono essere raggruppate prima della costruzione di ogni albero, questo metodo è in grado di velocizzare notevolmente l'addestramento, riducendone la complessità computazionale a O (n_osservazioni ; n_bins)[19].

[18] https://lorenzogovoni.com/cose-lalgoritmo-xgboost/

[19] https://ichi.pro/it/algoritmi-di-potenziamento-adaboost-gradient-boosting-xgb-light-gbm-e-catboost-254892816262346

- Esegue un numero massimo di split fino alla profondità massima specificata dal parametro *max_dept*, invece che utilizzare l'impurità come il Gradient Boosting, e successivamente inizia a "*potare l'albero all'indietro*", andando a rimuovere gli split oltre i quali non vi è alcun guadagno positivo di informazione; questo approccio migliora sensibilmente le prestazioni di calcolo.

In generale, se si escludono i problemi in cui si deve prevedere dati non strutturati (come immagini o testo), rispetto ai quali le reti neurali riescono ad offrire ottime prestazioni, per tutti gli altri casi in cui si ha dati strutturali (tabulari di piccole, medie o grandi dimensioni) gli algoritmi basati sugli alberi decisionali sono quelli considerati maggiormente prestanti. Nonostante questo, quando si tratta un problema di Machine Learning è necessario testare più algoritmi al fine di produrre un'analisi più accurata ed esaustiva. Inoltre, non è sufficiente limitarsi a scegliere l'algoritmo giusto; infatti, il processo di gran lunga più difficile è quello riguardante l'individuazione della migliore configurazione dei parametri rispetto al dataset di partenza e all'algoritmo scelto. Per i motivi sopracitati, in questo progetto di tesi sono stati sviluppati tre diversi modelli di Machine Learning profondamente diversi tra loro e con diversi setting degli iperparametri: Regressione Logistica, Rete Neurale e XGBoost.

ASPETTI COMUNI DEI MODELLI

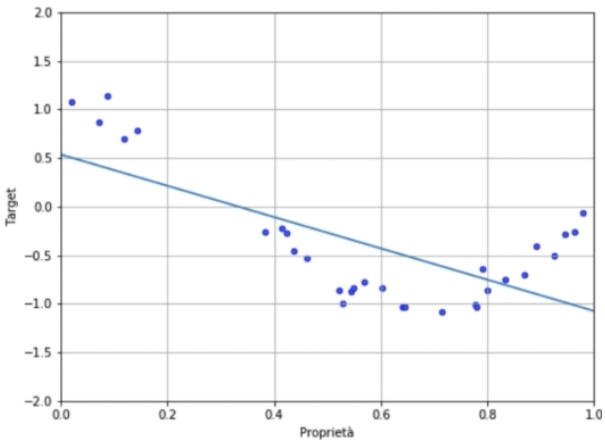
Overfitting: il Problema

Prima di porre l'attenzione su uno dei principali problemi che può affligge un generale modello di Machine Learning, è necessario introdurre due concetti fondamentali: *Bias* e *Varianza*.

Il Bias è una metrica che indica quanto sono distanti le previsioni fornite dal modello dalla target corretta; misura l'errore sistematico del modello, quello che non dipende dalla casualità o dai dati immessi in input. La Varianza misura la differenza nelle predizioni dei modelli costruiti utilizzando sezioni diverse del dataset, quindi ci indica la sensibilità del modello alla casualità dei dati del training set.

Le due metriche sono molto correlate tra loro tanto che quando aumenta una sistematicamente diminuisce l'altra (trade off); lo scopo è quello di trovare la corretta combinazione che renda entrambe le più piccole possibili. Per comprendere al meglio, si osserva la struttura di una regressione lineare semplice:

$$y = b + w_1 X_1$$



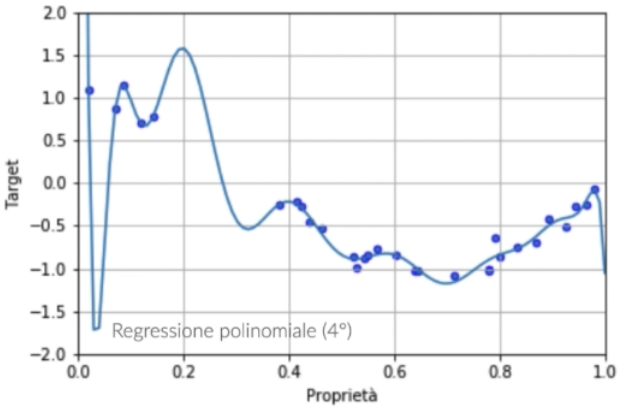
la varianza risulta bassa in quanto riaddestrando il modello con altre parti del dataset si otterrebbe in ogni caso un modello molto simile. Il bias, invece, è molto alto perché mediamente la differenza tra il

valore predetto dal modello e quello corretto è molto alta (distanza tra la retta e i punti).

Ne consegue che vi è un chiaro errore: il modello è troppo semplicistico e generico per riuscire a catturare tutti i pattern interni ai dati; questo problema è chiamato *Underfitting*.

Al contrario, complicando il modello si può arrivare alla seguente situazione:

$$y = b + w_1X + w_2X^2 + w_3X^3 + w_4X^4$$



si utilizza un modello di regressione di quarto grado. In questo caso il bias è molto basso, quasi nullo, dato che il modello è molto preciso nell'effettuare previsioni riguardo il target, mentre la varianza sarà troppo alta in quanto il modello approssima eccessivamente bene i dati di training.

Questo porta a un problema opposto al precedente: *Overfitting*, ovvero il rischio di ottenere tutt'altro modello (con altre performance) nel momento in cui si dovesse eseguire nuovamente l'addestramento con una qualsiasi altra parte del dataset. Dunque, un modello soffre di overfitting quando, invece che apprendere dai dati, li ha memorizzati; a causa della sua elevata complessità, il modello si lega eccessivamente ai dati con i quali si è addestrato e così non riesce a generalizzare su nuovi dati ciò che ha "appreso".

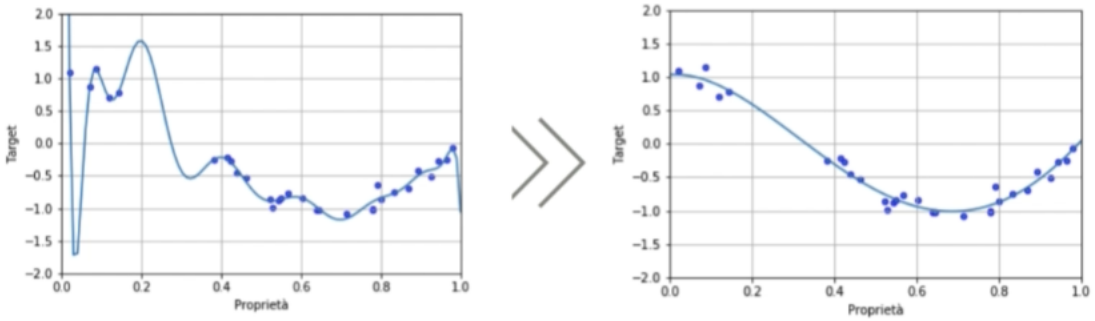
Individuare la presenza o meno di questo tipo di problema non è complesso, infatti è sufficiente confrontare, al termine della fase di addestramento, l'errore risultante sul test set e quello sul training set; se il primo è significativamente maggiore del secondo allora vi è problema di overfitting.

La causa principale è rappresentata dalla mancanza di sufficienti osservazioni per l'addestramento oppure quando si cerca di costruire una rete neurale eccessivamente complessa. La soluzione più immediata è quella di utilizzare delle regolazioni.

Tecniche di Regolarizzazione

L1 & L2

La regolarizzazione consiste nell'attenuare tutti quei coefficienti del modello che durante la fase di addestramento assumono entità troppo grande al fine di ridurne la varianza e, di conseguenza, la complessità.



Ciò farà apparire tutto più regolare e meno sensibile alle variazioni che potrebbero scaturire da un legame eccessivo dei dati di training. Ne esistono di due tipi: la regolarizzazione L1 e L2; entrambe aggiungono una penalizzazione per pesi molto grandi alla funzione di costo durante la fase di addestramento.

Con il regolatore L2 (weight decay) il fattore penalizzante è la somma quadratica dei pesi:

$$cost(w,b) + \boldsymbol{\lambda} \sum_{\boldsymbol{j=1}}^{\boldsymbol{M}} \boldsymbol{w_j^2}$$

dove λ è detto parametro della regolarizzazione; è un iperparametro che definisce il peso che avrà la regolarizzazione (λ alto significa grande importanza). Tale aggiustamento costringerà il Gradient Descent a cercare il minimo di questa nuova funzione per valori dei coefficienti più piccoli.

Nella regolarizzazione L1 il termine da aggiungere alla funzione di costo è, invece, la somma del valore assoluto dei pesi:

$$cost(w, b) + \boldsymbol{\lambda} \sum_{\boldsymbol{j=1}}^{\boldsymbol{M}} |\boldsymbol{w_j}|$$

Tale regolarizzazione risulta decisamente più intensa della L2 in quanto il nuovo termine porta il peso delle proprietà meno importanti direttamente a *zero*. Dato che una feature con peso zero viene totalmente esclusa dal modello, la regolarizzazione L1 risulta particolarmente adatta per effettuare una selezione delle features (*features selection*):

$$w = [w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9]$$

$$w = [w_1, w_2, 0,0,0,0, w_7, 0,0]$$

$$y = b + w_1 X_1 + w_2 X_2 + w_7 X_7$$

La regolarizzazione L2 fornisce per la maggioranza dei casi pratici risultati migliori della L1; nonostante ciò risulta una buona tecnica utilizzarle entrambe sfruttando eventualmente due diversi valori di λ:

$$cost(w, b) + \boldsymbol{\lambda_1} \sum_{\boldsymbol{j=1}}^{\boldsymbol{M}} |\boldsymbol{w_j}| + \boldsymbol{\lambda_2} \sum_{\boldsymbol{j=1}}^{\boldsymbol{M}} \boldsymbol{w_j^2}$$

Il confronto tra λ_1 e λ_2 indicherà quale delle due regolarizzazioni deve avere maggiore importanza.

Essendo λ un iperparametro è necessario ottimizzare il suo valore; se λ=0 il termine di regolarizzazione è uguale a zero e la regolarizzazione non ha alcun

effetto sulla funzione di costo, dunque se il modello soffriva di overfitting continuerà a soffrirne. Viceversa, con un valore di λ troppo elevato (tendente all'infinito) il peso della regolarizzazione sarà eccessivo e la maggior parte dei coefficienti sarà ridotta a zero (se λ_1 dovesse avere un peso maggiore di λ_2); si avrà un problema di underfitting.

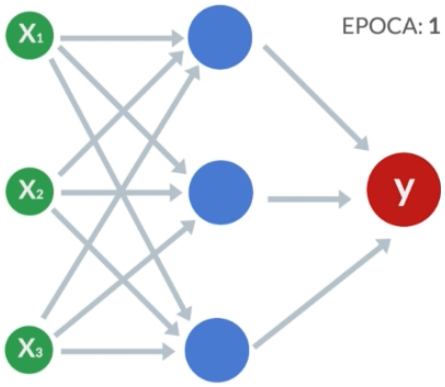
La valore ottimale è da ricercarsi in un range di potenze di 10, compreso tra:

$$10^{-4} \leq \lambda \leq 10$$

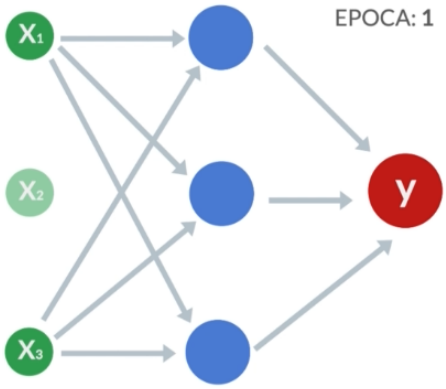
Dropout

A differenza delle due tecniche viste precedentemente, il dropout è una tecnica di regolarizzazione specifica per le reti neurali che consiste nel rimuovere un certo numero di nodi in uno strato della rete ad ogni iterazione della fase di addestramento; dunque, i nodi eliminati in una determinata epoca vengono selezionati casualmente ad ogni iterazione e non saranno presi in considerazione nella forward propagation e neanche nella back propagation.

Si considera un RNA con un unico strato nascosto:



Si vuole applicare il dropout al primo strato; si inizializza i coefficienti (pesi e bias) ad un valore casuale e, prima di avviare la prima epoca della fase di addestramento, si selezionano casualmente un nudo da rimuovere; viene selezionato il nodo X_2.



Con la rimozione del nodo X_2 devono essere eliminate anche tutte le connessioni ad esso associate.

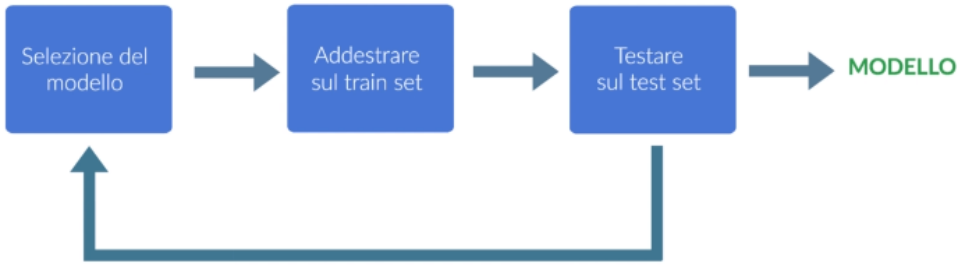
Si esegue la forward propagation per calcolare l'errore e successivamente lo si propaga all'indietro con la back propagation al fine di aggiornare i coefficienti dei nodi ancora attivi. Al termine dell'epoca il nodo eliminato viene ripristinato insieme alle sue connessioni e si procede con l'eliminazione casuale di un altro nodo ripetendo le operazioni precedenti per un numero determinato di epoche. In questo modo è come se si fossero addestrati più modelli con lo scopo successivamente di metterli insieme e creare un modello collettivo, replicando così l'apprendimento ensemble descritto nel capitolo precedente.

Il dropout è un metodo molto recente (2014) e può essere estremamente efficacie in quanto, durante la fase di addestramento, alcuni nodi della rete tendono a farsi carico degli errori commessi da altri nodi (*co-adaptations*); i coefficienti dei nodi che portano l'errore tenderanno ad azzerarsi mentre gli altri a crescere e questa condiziona porta inevitabilmente ad un problema di overfitting dato che il modello si legherà molto ai nodi con coefficienti troppo grandi. Al contrario, applicando il dropout vengono rimossi i nodi di supporto e di conseguenza la rete si ritroverà forzata a ottimizzare quelli rimanenti; infatti, quando vengono eliminati tutti i nodi che si sono fatti carico dell'errore, la rete dovrà necessariamente cercare di minimizzare l'errore utilizzando unicamente i nodi rimanenti.

Dunque, tutte le tecniche di regolarizzazione mirano a mantenere i parametri del modello vicini a zero, impedendo che ciascuno di essi "apprenda troppo"; in questo modo, garantiscono che il modello rimanga il più bilanciato possibile. Tuttavia, nel momento in cui il modello non risulta in grado di apprendere abbastanza dai dati di training, sarà necessario ridurre tali penalizzazioni.

Tecniche di Validazione

Si ponga l'attenzione sul seguente processo:



Selezionare un determinato modello, eseguire la fase di addestramento sul training set, testare il modello sul set di test e, in caso di risultati soddisfacenti, considerare tale modello definitivo; tuttavia, ogni modello ha diversi iperparametri da ottimizzare, quindi si potrebbe pensare che la procedura corretta sia quella di selezionare un modello con degli iperparametri settati in un certo modo, eseguire l'addestramento, testare il modello e, in caso di risultati deludenti sul test set, tornare alla selezione del modello cambiando gli iperparametri, ripetendo tale processo fin quando i risultati sul test non saranno soddisfacenti. Tale procedura è profondamente sbagliata; l'utilizzo del test set per la fase di *model selection* e del *hyperparameters tuning* è un errore tanto comune quanto grave in quanto il set di addestramento non è necessariamente l'unica parte del dataset che può portare all'overfitting;

infatti, addestrando uno stesso modello con molte configurazioni diverse alla ricerca della combinazione degli iperparametri migliore e testandolo sempre sullo stesso test set vi è il forte rischio che il modello, alla lunga, inizi a considerare il set di test come parte integrante del training set; questa particolare situazione scaturisce una condizione di *overfitting sul set di test*. Per ovviare a questo problema sono state sviluppate alcune tecniche di validazione; vengono descritte le due più utilizzate.

Validation Set

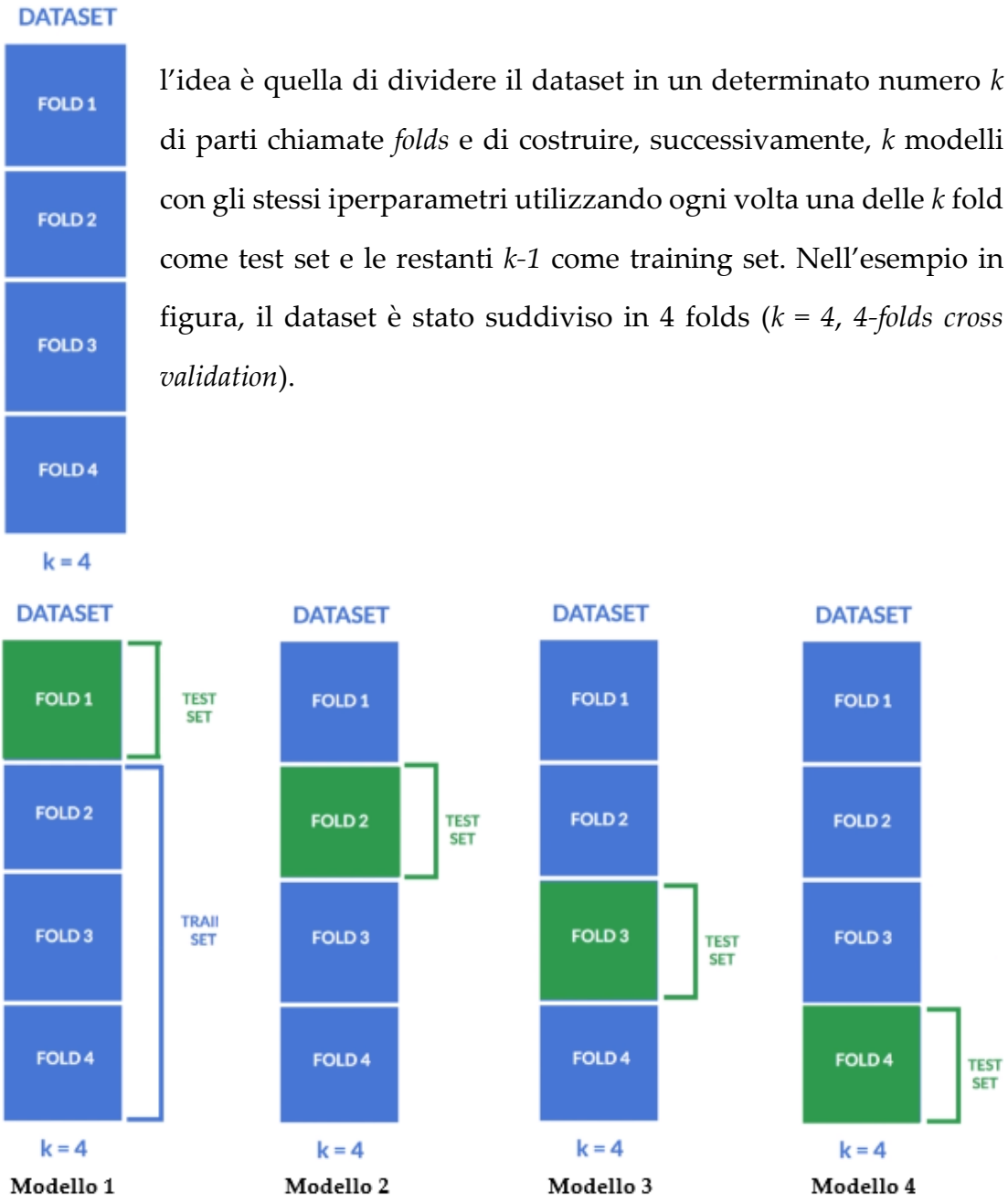
La più semplice e intuitiva è estrarre dal dataset un *terzo set* da utilizzare esclusivamente per validare il modello, ovvero per trovare il migliore setting degli iperparametri; questo set viene chiamato set di validazione (*validation set*); attraverso questa prima tecnica il dataset, dunque, sarà diviso in 3 distinti set:

il set di addestramento, contenente le osservazioni utilizzate per addestrare la rete, il set di validazione impiegato per ottimizzare gli iperparametri ed evitare l'overfitting e il set di test utilizzato esclusivamente alla fine (dopo aver selezionato il modello migliore) per verificare la capacità di generalizzazione della rete.

Tuttavia, la realizzazione di un nuovo set (validation) va inevitabilmente a ridurre il numero di osservazioni presenti nel training set; un ulteriore problema è il fatto che il modello potrebbe dipendere eccessivamente da come è stato eseguito lo split dei tre subset.

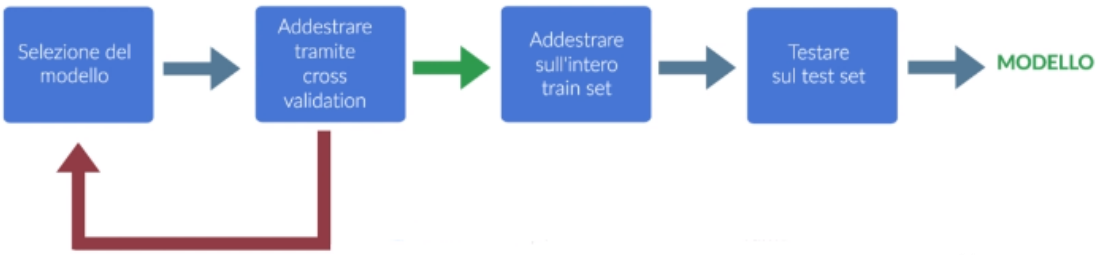
K-Fold Cross Validation

Alla luce degli evidenti problemi riscontrati nell'avere un validation set "statico", è stata sviluppata una migliore soluzione per verificare la qualità di un modello di Machine Learning denominata *K-Fold Cross Validation* (Validazione Incrociata).



l'idea è quella di dividere il dataset in un determinato numero *k* di parti chiamate *folds* e di costruire, successivamente, *k* modelli con gli stessi iperparametri utilizzando ogni volta una delle *k* fold come test set e le restanti *k-1* come training set. Nell'esempio in figura, il dataset è stato suddiviso in 4 folds (*k = 4*, *4-folds cross validation*).

Si ottengono 4 modelli per 4 folds; ogni modello, quindi, è identico all'altro in termini sia di tipologia (ad esempio, regressione logistica, rete neurale, albero decisionale) sia di settaggi degli iperparametri, differenziando esclusivamente per il train e test set che viene utilizzato. Per ognuno dei modelli se ne calcola l'errore ottenendo, attraverso la loro media, l'errore complessivo del modello.



La pratica migliore è utilizzare la cross validation invece che il set di validazione e tenere il set di test per la valutazione finale, nella quale il modello sarà testato con dati totalmente sconosciuti.

Il numero di folds da utilizzare varia a seconda della numerosità del dataset; in generale, $k = 10$ è il valore che meglio si adatta a ogni tipo di dataset. In presenza di un dataset molto grande o molto piccolo è consigliato un valore di k rispettivamente più piccolo ($k = 5$) o più grande ($k = 20$). Infatti, con un numero più grande di folds il numero di osservazioni per l'addestramento sarà maggiore e il bias tende a diminuire (rischio di overfitting), mentre con k piccolo il bias tende ad aumentare (rischio di underfitting); inoltre, in quest'ultimo caso la cross validation sarà più veloce da eseguire in quanto si dovrà addestrare un numero minore di modelli.

Tecniche di Ottimizzazione degli Iperparametri

Nonostante il Deep Learning sia un campo in continua crescita, ci sono alcuni aspetti pratici che rimangono una scatola nera come, ad esempio, l'ottimizzazione degli iperparametri (*hyperparameter tuning*). Si ricorda che gli iperparametri (learning rate, batch_size, dropout e gli altri descritti in precedenza) sono tutte quelle variabili che definiscono l'architettura di un modello e, come tali, devono essere necessariamente definite prima della fase di addestramento; controllando direttamente il comportamento dell'algoritmo di addestramento, questi risultano avere un impatto determinante sulle prestazioni del modello finale. Per ottenere il modello "perfetto", ognuno di essi dovrebbe essere accuratamente ottimizzato ma, a causa della loro numerosità, questo risulta estremamente difficile in particolar modo se si tratta di una rete neurale o modelli ad albero.

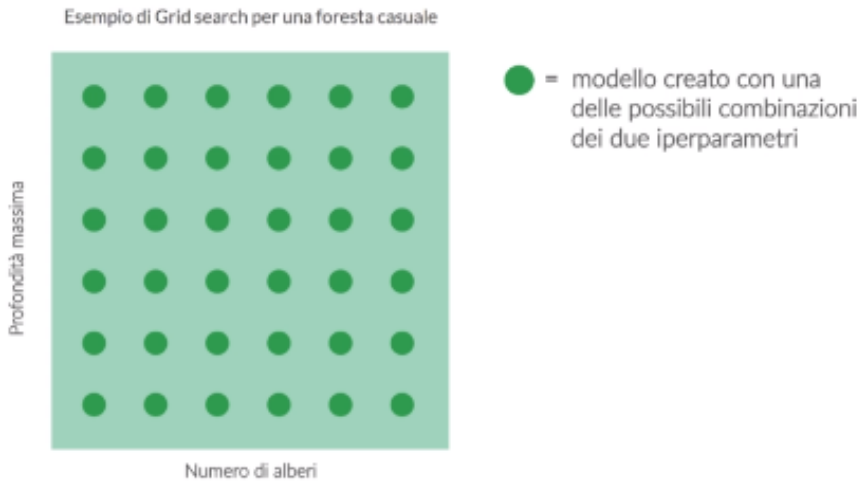
Manual Search

La fase di validazione verifica la robustezza di un modello settato con determinati parametri; lo step precedente è rappresentato dall'ottimizzazione della scelta di tali iperparametri:

Quella in figura è la procedura sviluppata fino ad ora: scegliere i valori degli iperparametri, addestrare il modello, verificarlo attraverso le tecniche di validazione e, in presenza di scarsi risultati, impostare manualmente altri valori; tale metodo, chiamato *Manual Search*, si basa esclusivamente sull'esperienza pregressa maturata dall'analista che permette di avvicinarsi approssimativamente ai parametri ottimali di un modello per una determinata tipologia di problema.

Grid Search

Esistono tuttavia altri approcci all'hyperparameter tuning che, al contrario, sono di tipo sperimentale; di seguito viene presentato un metodo tra i più utilizzati: la *Grid Search* (approccio *a forza bruta*):



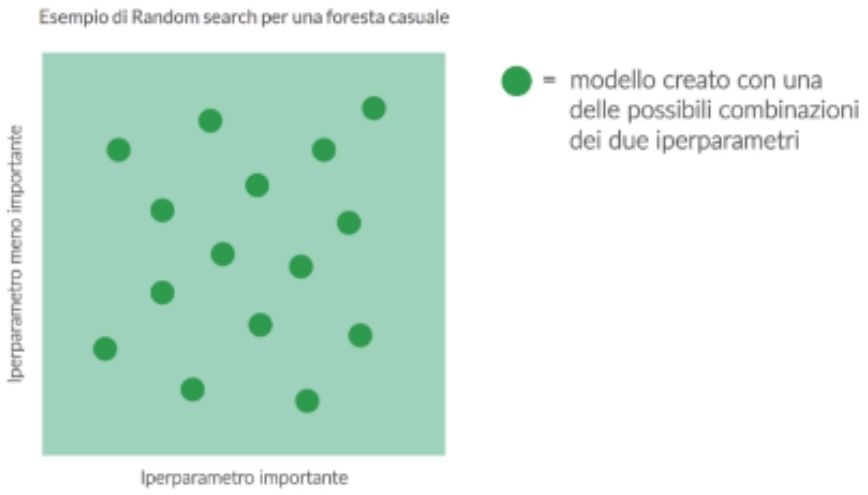
vengono scelti alcuni valori per diversi iperparametri da ottimizzare e la Grid Search proverà tutte le combinazioni possibili di questi, ovvero addestrerà un modello per ogni possibile combinazione e, infine, ritornerà quello migliore misurando le prestazioni mediante la tecnica della Cross Validation. L'esempio in figura mostra la situazione in cui un programmatore ha deciso

di voler ottimizzare 2 iperparametri (*numero di alberi* e *profondità massima*) di un modello ad albero decisionale.

Nonostante questa sia la tecnica sperimentale più utilizzata per il tuning, effettuare una ricerca esaustiva e completa di tutti i possibili valori che possono assumere gli iperparametri è estremamente dispendioso in termini di tempo e risorse di calcolo.

Random Search

Un'altra tecnica è la *Random Search* che, al contrario, cerca di ottimizzare gli iperparametri selezionandoli casualmente da una distribuzione di valori:



Dato che normalmente gli iperparametri non hanno tutti la stessa importanza all'interno del processo di addestramento, sebbene sia possibile che la Random Search non offra un risultato esaustivo come la Grid Search, spesso la prima riesce a trovare la migliore combinazione e in un tempo decisamente inferiore a quello che avrebbe impiegato la Grid Search.

Di contro questa tecnica ha il fatto che, essendo un approccio casuale, ogni volta che viene rieseguita è molto probabile che offra sempre nuove

combinazioni; per questo motivo è consigliabile eseguire, successivamente alla fase di ottimizzazione, una separata cross validation con gli iperparametri indicati dalla Random Search al fine di verificarne la validità e robustezza.

Dunque, si riassumono gli step seguiti per il caso pratico di questo lavoro di tesi:

I. Fase di Features Selection;
II. Split del dataset in train e test set (approfondito nel prossimo capitolo);
III. Scaler del dataset (approfondito nel prossimo capitolo);
IV. Selezione del modello;
V. Esecuzione della Random Search per l'ottimizzazione degli iperparametri;
VI. Ripetizione, se necessario, del passo precedente fino al raggiungimento di risultati soddisfacenti;
VII. Esecuzione dell'addestramento utilizzando la Cross Validation sui K-Folds, al fine di validare il modello settato con gli iperparametri scaturiti dalla Random Search;
VIII. Ripetizione dell'addestramento (questa volta) sull'intero training set;
IX. Verificare il modello sul test set;
X. In caso di risultati accettabili il modello sarà ritenuto definitivo.

PROGETTO: ANALISI CREDIT SCORING

Selezione del Settore Economico

L'analisi di credit scoring è il particolare processo attraverso il quale le banche o altri istituti finanziari attribuiscono un certo punteggio (*score*) a un individuo o società al fine di determinare il loro grado di affidabilità creditizia (*creditworthiness*). La valutazione del cliente, e quindi l'assegnazione di un certo score, scaturisce in modo automatico dall'applicazione dei vari modelli di Machine Learning come la regressione logistica o la rete neurale artificiale descritti nel capitolo precedente. Dunque, il credit scoring rappresenta uno dei principali metodi utilizzati per stimare il rischio di credito pendente su un soggetto o impresa.

Il progetto di tesi, con i vari modelli che seguiranno, ha come obiettivo quello di prevedere, esclusivamente sulla base dei principali indici di bilancio, se una società andrà o meno in stato di insolvenza dato che, come già espresso, si considera "rischio" il solo evento default/non default. Nei casi reali il rischio è stimato prendendo in considerazioni non solo i fattori quantitativi ma anche quelli di natura qualitativa come la forma giuridica, il modo di tenere la contabilità oppure il settore in cui opera. È proprio quest'ultimo uno dei primi fattori che deve essere accuratamente selezionato; infatti, è necessario poter contare su un settore economico composto da un discreto numero di imprese e, preferibilmente, con un buon rapporto tra società in default e sane.

Tra i vari settori economici la scelta è ricaduta su quello Metallurgico; esso risulta uno dei più importanti in Italia influenzando direttamente l'intero settore secondario del paese. Inoltre, l'industria metallurgica sorregge un

indotto di svariati miliardi di euro in quanto i prodotti da essa derivanti vengono utilizzati in molte altre aree economiche come l'automotive, costruzioni e edilizia. Il settore è identificato univocamente con il codice ATECO 24; in tale divisione sono comprese le attività di fusione e/o affinazione di metalli ferrosi e non ferrosi a partire da minerali, lingotti metallici o rottame metallico, con tecniche elettrometallurgiche ed altre tecniche metallurgiche[20]. Rientra, inoltre, la produzione di leghe e superleghe di metalli, con l'aggiunta nei metalli puri di altri elementi chimici. I prodotti ottenuti dalla fusione e dalla affinazione, generalmente in forma di lingotti vengono trasformati con processo di laminazione, trafilatura ed estrusione in lamiere, nastri, barre, tondi o vergella e in forma fusa, per realizzare pezzi di fonderia e altri prodotti metallici. All'interno della divisione 24 sono presenti 3143 imprese suddivise tra i gruppi:

- 24.1 Siderurgia = Attività quali la riduzione diretta del minerale di ferro, la fabbricazione di ghisa grezza in forma liquida o solida, la conversione di ghisa grezza in acciaio, la fabbricazione di ferroleghe e di prodotti siderurgici;
- 24.2 Fabbricazione di tubi, condotti, Profilati cavi e relativi accessori in acciaio;
- 24.3 Fabbricazione di altri prodotti della prima trasformazione dell'acciaio;
- 24.4 Produzione di metalli di base preziosi e altri metalli non ferrosi, trattamento dei combustibili nucleari;

[20]http://www.codiciateco.it/metallurgia/C24#:~:text=24%20METALLURGIA,elettrometallurgiche%20ed%20altre%20tecniche%20metallurgiche

- 24.5 Fonderie = Fabbricazione di prodotti semilavorati e getti di vario genere che possono essere anche prodotti finiti, tramite fusione.

Data l'importanza di poter gestire un elevato numero di società si è deciso di inserire nell'analisi anche tutte quelle che operano nella divisione 25 e gruppo 5 (quindi 25.5), rappresentante le attività di fucinatura, imbutitura, stampaggio e profilatura dei metalli e la metallurgia delle polveri, che ha permesso di avere i dati di 4624 imprese totali.

Il campione: Creazione e Analisi

Dopo aver selezionato il settore oggetto di analisi, la fase successiva è l'acquisizione dei dati; questi sono stati ottenuti dalla Banca Dati "AIDA". Tramite il codice ATECO, da tale database sono stati selezionati e scaricati, per ognuna delle imprese operanti nella metallurgia, tutti i bilanci (Stato Patrimoniale e Conto economico) dal 2009 al 2018, in modo tale da disporre di una discreta mole di dati rappresentativa di 10 anni di serie storiche, e la loro anagrafica (anno per anno) costituita dalle seguenti informazioni:

- Anno del bilancio;
- Ragione sociale;
- Anno di costituzione;
- Luogo della sede legale: comune, provincia, regione;
- Codice ISTAT: comune, provincia, regione;
- Forma giuridica;
- Codice fiscale
- Bilancio consolidato/non consolidato
- Bilancio rispetta/non rispetta i IFRS
- Bilancio dettagliato/abbreviato

- Codice ATECO 2007
- Stato giuridico
- Impresa quotata/non quotata
- Eventuale procedura/cessazione
- Data di inizio della procedura/cessazione
- Numero dei dipendenti

Tutte le informazioni sopracitate sono state scaricate un anno alla volta, su file separati; dunque, all'inizio si è proceduto al download di tutti i bilanci e anagrafica del 2009, successivamente del 2010 fino ad arrivare a quelli del 2018. Dopo di che si sono uniti in un unico file Excel i 10 creati in precedenza e, con un ordinamento crescente rispetto all'anno e alla ragione sociale, si arriva a visualizzare per ogni impresa tutti i bilanci in serie storica (dal 2009 al 2018 oppure fino all'ultimo anno nel quale è disponibile il bilancio per un'impresa):

Anno	n osservazione	Num.	Ragione Sociale	Anno Costituzione	...	Procedura/ Cessazione	Data di inizio procedura/ cessazione	Dipendenti
2009	1	1	ILVA S.P.A.	08/02/1995		Stato di inso	28/01/2015	15743
2010	2	1	ILVA S.P.A.	08/02/1995		Stato di inso	28/01/2015	15066
2011	3	1	ILVA S.P.A.	08/02/1995		Stato di inso	28/01/2015	14790
2009	4	2	MARCEGAGLIA	27/06/1983		Fusione mec	26/01/2016	3631
2010	5	2	MARCEGAGLIA	27/06/1983		Fusione mec	26/01/2016	3591
2011	6	2	MARCEGAGLIA	27/06/1983		Fusione mec	26/01/2016	3608
2012	7	2	MARCEGAGLIA	27/06/1983		Fusione mec	26/01/2016	3694
2013	8	2	MARCEGAGLIA	27/06/1983		Fusione mec	26/01/2016	3698
2014	9	2	MARCEGAGLIA	27/06/1983		Fusione mec	26/01/2016	3646
2015	10	3	MARCEGAGLIA	29/05/2015				2734
2016	11	3	MARCEGAGLIA	29/05/2015				2738
2017	12	3	MARCEGAGLIA	29/05/2015				2741
2018	13	3	MARCEGAGLIA	29/05/2015				2604
2010	14	4	ITALPREZIOSI S	15/03/1984				15
2011	15	4	ITALPREZIOSI S	15/03/1984				23
2012	16	4	ITALPREZIOSI S	15/03/1984				32

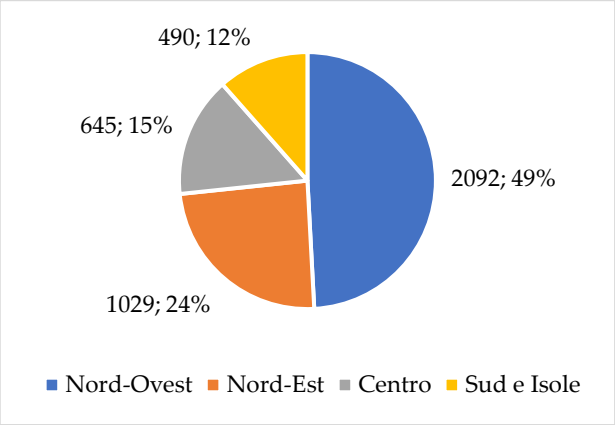
Figura 7: Foglio 1

ogni riga caratterizza un bilancio di una determinata società a un determinato anno (elencati in ordine crescente) mentre le colonne rappresentano in ordine:

le informazioni anagrafiche, lo Stato Patrimoniale (Attivo, Passivo) e il Conto Economico.

Dalle 4624 imprese inizialmente selezionate sono state rimosse 368, ovvero tutte quelle che presentavano un Attivo Patrimoniale nullo per ogni anno di bilancio, arrivando in questo modo a un numero totale di 4256; di queste, oltre l'80% (3526 imprese) hanno come forma giuridica una S.R.L., il 15,5% una S.P.A. -659 imprese-, mentre il restante è equamente suddiviso fra S.A.S., S.C.A.R.L.P.A., S.C.A.R.L., S.N.C. e Consorzio.

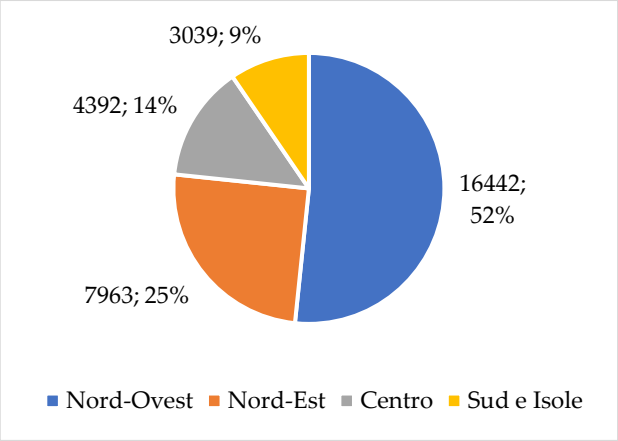
Di seguito viene mostrato come le imprese sono distribuite all'interno delle 4 macroaree italiane:



si nota facilmente come la maggior parte delle aziende metallurgiche (circa la metà del totale) abbiano sede legale nel Nord-Ovest del paese (Valle d'Aosta, Piemonte, Liguria e Lombardia). La sola Lombardia, la quale risulta essere la regione maggiormente popolata, registra 1592 imprese, oltre il 37% del totale. Veneto, Piemonte e Emilia-Romagna sono anch'esse regioni con una discreta presenza avendo ognuna un numero di aziende metallurgiche al di sopra delle 400 unità, mentre il Centro, Sud e le isole hanno una popolosità significativamente inferiore.

Dal punto di vista delle osservazioni, ovvero il campione che sarà fornito in input ai successivi modelli, queste sono pari a 31836, dunque è come se per ognuna delle 4256 imprese mediamente fossero disponibili circa 7,5 anni di bilancio ($^{31836}/_{4256}$); tale campione presenta valori medi (sulle 31836

osservazioni) dei ricavi delle vendite, costo delle materie prime e utile netto rispettivamente di 19,2 miliardi €, 13,5 miliardi€ e 38 mila € con circa 40 dipendenti per osservazione.



La distribuzione delle singole osservazioni per area ripercorre grossomodo con le stesse proporzioni quella del numero delle imprese.

Il dettaglio di ogni regione è rappresentato dalla tabella sottostante:

Regione	# **imprese**	%	# **esempi**	%
Toscana	179	4,21%	1332	4,18%
Lombardia	1592	37,41%	12717	39,95%
Umbria	51	1,20%	344	1,08%
Friuli-Venezia Giulia	104	2,44%	862	2,71%
Trentino-Alto Adige	44	1,03%	261	0,82%
Emilia-Romagna	413	9,70%	3237	10,17%
Lazio	169	3,97%	1040	3,27%
Piemonte	458	10,76%	3383	10,63%
Marche	160	3,76%	1138	3,57%
Veneto	468	11,00%	3603	11,32%
Valle d'Aosta	6	0,14%	55	0,17%
Liguria	36	0,85%	287	0,90%
Basilicata	19	0,45%	128	0,40%
Campania	253	5,94%	1518	4,77%
Sicilia	92	2,16%	554	1,74%

Elaborazione dei dati

Creazione della Variabile Target

Come viene esposto nella sezione riguardante l'addestramento di una Rete Neurale Artificiale, per ogni osservazione data in input ai modelli è necessario fornire anche la sua target esatta, nel caso in esame un valore binario 0/1, in modo tale da indicare alla rete quale output dovrebbe tirare fuori. In particolare, avere una target uguale a uno in una generica osservazione *i* significa che il bilancio risultante nella riga *i* di una certa società in un certo anno è un bilancio di una società che è anomala; al contrario, con una target uguale a zero la società in quel preciso anno risulta sana. Per determinare quali casi rientrano nella dicitura "anomala" viene utilizzata una struttura a 4 flag, ognuno dei quali indica le condizioni per le quali una società in un dato anno è in un certo stato di salute:

- **Flag = 0 (Società sana)**
- **Flag = 1 (Società anomala):**
    - Concordato preventivo
    - Fallimento
    - Amministrazione giudiziaria
    - Accordo di ristrutturazione dei debiti
    - Chiusura del fallimento
    - Liquidazione giudiziaria
    - Stato di insolvenza
    - Sequestro giudiziario
    - Concordato fallimentare
    - Amministrazione controllata
    - Cancellazione per comunicazione piano di riparto

- Amministrazione straordinaria
- Chiusura per fallimento o liquidazione
- Decreto cancellazione tribunale
- Liquidazione coatta amministrativa
- Scioglimento per atto dell'autorità
- Sequestro conservativo di quote
- Bancarotta

- **Flag = 2 (Società sana in condizioni particolari):**
  - Liquidazione volontaria
  - Scioglimento e liquidazione
  - Scioglimento
  - Chiusura della liquidazione
  - Chiusura dell'unità locale
  - Cessazione di ogni attività
  - Cancellata d'ufficio ai sensi art. 2490 c.c. (bilancio in liquidazione)
  - Liquidazione
  - Scioglimento e messa in liquidazione
  - Chiusura per liquidazione
  - Scioglimento senza messa in liquidazione
  - Cessazione delle attività nella provincia
  - Cessazione d'ufficio
- **Flag = 3 (Società sana in condizioni particolari):**
  - Fusione mediante incorporazione in altra società
  - Scissione
  - Trasferimento sede all'estero
  - Fusione mediante costituzione di nuova società

- Cessione azienda

- **Flag = 4 (Società sana in condizioni particolari):**
  - Cessata cancellata dal registro impresa
  - Trasferimento in altra provincia

A tal riguardo, è fondamentale la colonna *"Procedura/Cessazione"* della *figura 1* precedente in quanto segnala a quale delle condizioni sopra elencate appartiene un'azienda in un dato anno; quindi, in un secondo foglio Excel di appoggio si procede ad associare ad ogni riga (osservazione) il corrispettivo flag (colonna *E*):

	A	B	C	D	E	F	G
1	Anno	Ragione Sociale	Procedura/Cessazione	Data di inizio procedura/cessazione	**Associazione FLAG 01234**	**SOCIETÀ - FLAG 1**	**ANNO - FLAG 1**
2	2009	ILVA S.P.A.	Stato di insolvenza	28/01/2015	1	1	0
3	2010	ILVA S.P.A.	Stato di insolvenza	28/01/2015	1	1	0
4	2011	ILVA S.P.A.	Stato di insolvenza	28/01/2015	1	1	0
5	2009	MARCEGAGLIA S.R.L.	Fusione mediante incorpor	26/01/2016	3	0	0
6	2010	MARCEGAGLIA S.R.L.	Fusione mediante incorpor	26/01/2016	3	0	0
7	2011	MARCEGAGLIA S.R.L.	Fusione mediante incorpor	26/01/2016	3	0	0
8	2012	MARCEGAGLIA S.R.L.	Fusione mediante incorpor	26/01/2016	3	0	0
9	2013	MARCEGAGLIA S.R.L.	Fusione mediante incorpor	26/01/2016	3	0	0
10	2014	MARCEGAGLIA S.R.L.	Fusione mediante incorpor	26/01/2016	3	0	0

...

67	2009	LUCCHINI S.P.A.	Trasferimento in altra prov	30/04/2013	3	0	0
68	2010	LUCCHINI S.P.A.	Trasferimento in altra prov	30/04/2013	3	0	0
69	2011	LUCCHINI S.P.A.	Trasferimento in altra prov	30/04/2013	3	0	0
70	2009	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0
71	2010	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0
72	2011	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0
73	2012	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0
74	2013	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0
75	2014	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0
76	2015	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0
77	2016	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	1
78	2017	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0
79	2018	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0

Figura 8: Formazione dei FLAG

La colonna *"Associazione FLAG 01234 (E)"* rappresenta la base per andare a determinare quella che sarà la target (0/1) effettiva di ogni osservazione.

Inizialmente, si decise di considerare anomala una società che in un certo anno presentasse nella colonna *E* la sola associazione con il flag 1; a questo punto la *variabile target* è stata costruita in due diversi modi:

- per Società *(colonna F)*: vengono poste a 1 tutte le righe delle società che in un qualsiasi anno tra il 2009 e il 2018 sono risultate anomale *(colonna E = 1)*, indipendentemente dall'anno di inizio procedura. È chiarificatore il caso della società Carlo Colombo S.p.A. in cui si sono messe a 1 tutte le righe (da 70 a 79) anche se l'anno effettivo del default è stato il 2016.
- per Anno *(colonna G)*: viene posto 1 solo nelle righe in cui la colonna *E* risulta uguale a 1 e l'anno corrente del bilancio coincide con l'anno di inizio procedura; prendendo in esame ancora l'esempio della Carlo Colombo S.p.A., si vede come l'1 sia in corrispondenza del 2016.

Con questa procedura si sono costruite due variabili target che rappresentano due visioni completamente opposte di quello che sarà l'output di un modello di credit scoring a scopo predittivo. Al fine di comprendere meglio come la scelta di una target -rispetto ad un'altra- influenza l'output del modello, si approfondisce la situazione della Carlo Colombo S.p.A.:

	A	B	C	D	E	F	G	H
1	Anno	Ragione Sociale	Procedura/Cessazione	Data di inizio procedura/cessazione	**Associazione FLAG 01234**	**SOCIETÀ - FLAG 1**	**ANNO - FLAG 1**	X
70	2009	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0	23385,466
71	2010	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0	10618,122
72	2011	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0	9569,179
73	2012	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0	8997,591
74	2013	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0	8261,16
75	2014	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0	184,499
76	2015	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0	87,068
77	2016	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	1	0
78	2017	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0	0
79	2018	CARLO COLOMBO S.P.A.	Accordo di ristrutturazione	14/09/2016	1	1	0	0

Figura 9

Data una generica variabile X, si supponga di inserirla in input come feature del modello; i suoi valori diminuiscono costantemente durante gli anni fino ad arrivare a 0 nel 2016, ovvero nell'anno dell'effettivo default. A questo punto se si utilizza come target quella *anno* e si considera tali valori all'interno, ad esempio, di un classico modello di regressione logistica, questi rappresenteranno la variabile X di una società che non fallisce fino all'anno 2016, in quanto si ha 0 per tutti gli anni dal 2009 al 2015. Questo produce una riduzione dell'effetto della variabile X e quindi il relativo β_X avrà una stima al ribasso (sarà più basso del dovuto). Al contrario, la target *società* scaturisce sul modello l'effetto contrario, ovvero una sovrastima del coefficiente β_X dovuta al fatto che da troppi anni prima si invia un "segnale di default" con l'*1* posto ad ogni riga di bilancio dal 2009.

Si arriva alla conclusione che nessuna delle due target è in realtà quella ottima da considerare ma, al tempo stesso, individuare quella "perfetta" richiederebbe uno studio approfondito di ogni bilancio di ogni impresa; dunque, per semplicità si è deciso di tenere e di fornire entrambe ai successivi modelli al fine di valutarne la differenza nelle performance.

Come espresso a inizio capitolo, oltre alla numerosità del campione in esame è anche importante disporre, all'interno di esso, di un buon rapporto anomale/sane in modo tale da rendere efficace l'analisi di scoring. La target costruita seguendo il principio *anno* presenta un problema di questo tipo in quanto conta un numero di *1* pari a 61, decisamente troppo pochi rispetto alla numerosità del campione. Per questo motivo si è deciso di inserire nel computo delle anomale non solo le società con bilanci associati con il *flag = 1* ma anche quelle che registrano un'associazione con il *flag = 2*; dunque, le due colonne della target presenteranno un *1* sia quando la società è in stato di insolvenza (*flag = 1*) sia quando questa è in fase di liquidazione (*flag = 2*). In

ogni caso i due metodi di costruzione della target e i ragionamenti espressi in precedenza rimangono validi. Applicando tale modifica, la target *anno* e *società* si vedono incrementare significativamente il conto degli *1* arrivando rispettivamente a 160 e 4857; data la natura differente dei due metodi è assolutamente ragionevole avere un numero di *1* estremamente maggiore nella target *società* ma ciò non significa che i modelli che utilizzeranno questa target avranno automaticamente performance superiori.

Correzione degli Errori

Successivamente alla creazione delle due target definitive, queste si trasferiscono in un secondo foglio Excel accuratamente connesso al primo (*figure 1 – foglio 1*) in modo tale da avere automaticamente calcolate un set di colonne rappresentati i principali indici di bilancio, alcuni dei quali saranno le variabili indipendenti (features) per i modelli che seguiranno.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	n.osservazioni (sequenza dopo ordinamento date dei bilanci)	n.società	RAGIONE SOCIALE	anno di riferimento del bilancio	Bilancio cons/non cons	Bilancio ordinario /abbreviato	DEFAULT	FLAG continuità bil (1=interruz; 0=continuità)	FLAG bilancio inesistente (=1)	SOCIETÀ - FLAG 1	ANNO - FLAG 1	SOCIETÀ - FLAG 1 & 2	ANNO - FLAG 1&2
2	1	1	ILVA S.P.A.	2009	Non Cons.	Dettagliato		1	0	1	0	1	0
3	2	1	ILVA S.P.A.	2010	Non Cons.	Dettagliato		0	0	1	0	1	0
4	3	1	ILVA S.P.A.	2011	Non Cons.	Dettagliato		0	0	1	0	1	0
5	4	2	MARCEGA	2009	Non Cons.	Dettagliato		1	0	0	0	0	0
6	5	2	MARCEGA	2010	Non Cons.	Dettagliato		0	0	0	0	0	0
7	6	2	MARCEGA	2011	Non Cons.	Dettagliato		0	0	0	0	0	0
8	7	2	MARCEGA	2012	Non Cons.	Dettagliato		0	0	0	0	0	0
9	8	2	MARCEGA	2013	Non Cons.	Dettagliato		0	0	0	0	0	0
10	9	2	MARCEGA	2014	Non Cons.	Dettagliato		0	0	0	0	0	0
11	10	3	MARCEGA	2015	Non Cons.	Dettagliato		1	0	0	0	0	0
12	11	3	MARCEGA	2016	Non Cons.	Dettagliato		0	0	0	0	0	0
13	12	3	MARCEGA	2017	Non Cons.	Dettagliato		0	0	0	0	0	0
14	13	3	MARCEGA	2018	Non Cons.	Dettagliato		0	0	0	0	0	0
15	14	4	ITALPREZI	2010	Non Cons.	Dettagliato		1	0	0	0	0	0
16	15	4	ITALPREZI	2011	Non Cons.	Dettagliato		0	0	0	0	0	0
17	16	4	ITALPREZI	2012	Non Cons.	Dettagliato		0	0	0	0	0	0

...

…

CO	CP	CQ	CR	CS	CT	CU
INDICATORI O=ordinari; nulla=abbreviati	sviluppo-->	var% ricav	var% valor	var % EBIT	var% AN	var % debi
		35,12418	80,3628	7,161295	2,512058	3,849392
		30,95333	-6,30393	-69,3119	8,862908	17,98549
		ND	ND	ND	ND	ND
		51,59328	25,46634	44,80077	7,88312	9,349209
		14,776	8,128091	9,707949	15,75397	22,15553
		-3,70176	1,244319	2,942196	8,174119	10,60886
		-4,75332	-3,86231	-8,52312	0,502634	3,056041
		1,387953	5,63867	10,31249	12,2531	18,42857
		ND	ND	ND	ND	ND
		108,4127	139,6857	303,1539	-3,99428	-0,90885
		33,18179	4,09256	6,048387	0,01862	0,495014
		6,189306	-12,5807	-19,4338	-19,7599	-17,3655
		ND	ND	ND	ND	ND
		72,61716	114,9469	164,2289	19,79628	16,805
		48,66085	72,07111	79,8273	52,70899	50,34301

Figura 10: Foglio 2

In questo *foglio 2* sono emersi diversi *errori* scaturiti dal fatto che nel foglio 1 molti dati risultavano nulli (come, ad esempio, nella colonna dei ricavi dalle vendite); infatti, gli errori più comuni hanno riguardato divisioni per 0 (#DIV/0!) e argomenti nulli in operazioni logaritmiche (#NUM!). Per ovviare, si è deciso di settare a 1 tutti i valori nulli del foglio 1 che provocavano tali errori nel foglio 2; conseguentemente, l'aver posto diverse celle a un valore così basso ha fatto sorgere nel foglio 2 un altro problema di diversa natura, ovvero quello degli outliers (valori estremamente distanti dal resto dei dati raccolti) che potrebbero "sporcare" il modello. La soluzione è stata quella di effettuare un livellamento al 5° e al 95° percentile su tutti gli indici di bilancio del secondo foglio; dunque, si è calcolato il 5° e il 95° percentile per ogni indice e, in seguito, i valori risultanti minori del 5° o maggiori del 95° percentile sono stati settati rispettivamente al 5° e 95° percentile, al fine di rimuovere i valori troppo bassi e, soprattutto, quelli troppo alti.

Costruzione del Dataset Strutturato

Con le ultime operazioni di pulizia dei fogli 1 e 2, si procede con la *creazione del Dataset strutturato* il quale, richiamando il nome, avrà un precisa struttura: questo conterrà una target (*società* o *anno*) e un insieme di variabili indipendenti (features) che, come già espresso, rappresentano alcuni degli indici di bilancio calcolati nel foglio 2. In questo lavoro di tesi si è deciso di rimuovere dal dataset tutte le informazioni anagrafiche dei bilanci e, dunque, di procedere solamente con features di tipo numerico; queste sono raggruppabili in 3 distinte macroaree:

- Redditività: gli indici di redditività valutano la capacità di un'azienda di ottenere profitto; ne fanno parte *ROE*, *ROE ante imposte*, *ROA*, *EBITDA/Ricavi*, *EBIT/ricavi*, *Utile_corrente/ricavi*, *EBITDA/AN* e molti altri;
- Produttività: indicano se i fattori produttivi sono stati impiegati in modo efficiente; tra questi *Valore_Aggiunto_Operativo/Ricavi*, *Consumi/costi_operativi*, *servizi_esterni/costi_operativi*, *Ricavi/AN*, *costo_lavoro/costi_operativi*, *gg_magazzino*, *amm_materiali/costi_operativi*;
- Liquidità: *AC/PC*, *AC-mag/PC*, *Liq/PC*, *Liq/AN*, *Cap_Circ/AN*, *Patr_netto/AN*, *Patr_netto_tang/AN*, *Patr_netto/debiti_totali*, *OF/RIC*, *OFN/RIC*, *Autof_Lordo/AN (=cash flow/attivo)* sono evidentemente indicatori di liquidità che mostrano in quale misura le attività correnti della società riescono a coprire i suoi debiti a breve termine.

Dopo essere stato convertito in un file *csv*, l'intero dataset è stato trasferito all'interno di un'applicazione web, chiamata Jupyter Notebook, la quale consente di eseguire il codice *Python*, ovvero il linguaggio di programmazione che si è deciso di utilizzare per l'intera realizzazione dei modelli. Di seguito si

ripercorrono su Python le operazioni appena descritte. Attraverso la libreria "*pandas*", il file csv "*credit_anno*", contenente il dataset, verrà caricato in un *Dataframe* (una speciale struttura dati tabulare di pandas) chiamato *credit*; con la funzione *.head()* si visualizzano le prime 5 osservazioni del dataframe:

```
import pandas as pd
import numpy as np

pd.set_option('display.max_columns',None)

credit= pd.read_csv(r'C:\Users\utente\Desktop\credit_anno.csv', encoding='latin1', sep=';')
credit.head()
```

	n.osservazioni (sequenza dopo ordinamento date dei bilanci)	n.società	RAGIONE SOCIALE	anno di riferimento del bilancio	FLAG di sana in liquid + anomala	EBITDA/Ricavi	EBIT/ricavi	Utile corrente/ricavi	Risultato netto rettif/ricavi	Risul n rettil
0	1	1	ILVA S.P.A.	2009	0	-3.44	-13.47	-8.65	-13.16	-
1	2	1	ILVA S.P.A.	2010	0	2.72	-5.56	1.43	0.93	
2	3	1	ILVA S.P.A.	2011	0	0.64	-5.86	-6.01	-0.58	-
3	4	2	MARCEGAGLIA S.R.L.	2009	0	6.16	3.15	1.54	-0.33	-
4	5	2	MARCEGAGLIA S.R.L.	2010	0	5.88	3.03	2.48	0.78	

```
credit.shape #DIMENSIONE DELL'INTERO DATASET
```

```
(31836, 51)
```

Il dataset risulta avere 31836 osservazioni e 51 variabili iniziali (tra anagrafiche e di bilancio), compresa la target.

Il nome del file csv "*credit_anno*" deriva dal fatto che questo dataset ha come risposta la variabile target creata con la logica *anno* [*FLAG di sana in liquid + anomala*]; questa contiene:

```
credit['FLAG di sana in liquid + anomala'].value_counts()
```

```
0    31676
1      160
Name: FLAG di sana in liquid + anomala, dtype: int64
```

e dunque presenta le seguenti percentuali di default/no default:

```
count_no_def = len(credit[credit['FLAG di sana in liquid + anomala']==0])
count_def = len(credit[credit['FLAG di sana in liquid + anomala']==1])
pct_of_no_def = count_no_def/(count_no_def+count_def)
print("percentage of no default is", pct_of_no_def*100)
pct_of_def = count_def/(count_no_def+count_def)
print("percentage of default", pct_of_def*100)
```

```
percentage of no default is 99.49742429953513
percentage of default 0.5025757004648825
```

Si crea un nuovo dataframe "*credit_only*", composto dalle sole variabili "*anno di riferimento del bilancio*" e la target, che servirà unicamente per visualizzare il numero di default per ogni anno preso in considerazione:

```
#creo un nuovo dataframe per VISUALIZZARE I DATI
credit_only= credit[['anno di riferimento del bilancio','FLAG di sana in liquid + anomala']]
credit_only
```

	anno di riferimento del bilancio	FLAG di sana in liquid + anomala
0	2009	0
1	2010	0
2	2011	0
3	2009	0
4	2010	0
...	...	...
31831	2009	0
31832	2009	0
31833	2009	0
31834	2010	0
31835	2009	0

31836 rows × 2 columns

```
credit_only.groupby('anno di riferimento del bilancio').sum()
```

	FLAG di sana in liquid + anomala
anno di riferimento del bilancio	
2009	12
2010	12
2011	12
2012	15
2013	25
2014	25
2015	15
2016	15
2017	16
2018	13

delle 160 osservazioni anomale oltre il 30% si sono manifestate tra il 2013 e 2014, mentre egli altri anni tale situazione risulta molto uniforme e livellata.

A questo punto si rimuovono dal dataset originario tutte le variabili di tipo anagrafico, aggiornando così la variabile *credit*:

```
credit= credit.drop(['n.osservazioni (sequenza dopo ordinamento date dei bilanci)','n.società',
                     'RAGIONE SOCIALE','anno di riferimento del bilancio'], axis=1)
#OPURE:credit.drop('id', axis=1, inplace= True)
credit
```

	FLAG di sana in liquid + anomala	EBITDA/Ricavi	EBIT/ricavi	Utile corrente/ricavi	Risultato netto rettif/ricavi	Risultato netto rettif/AN	EBITDA/AN	ROA	ROE	ROE ante imposte	Val Agg Oper/Ricavi	Consumi/costi operativi	servizi esterni/cos operativ
0	0	-3.44	-13.47	-8.65	-13.16	-6.82	-1.78	-2.65	-19.40	-20.94	11.48	56.33	19.0
1	0	2.72	-5.56	1.43	0.93	0.64	1.86	2.08	1.82	2.11	15.32	65.67	15.4
2	0	0.64	-5.86	-6.01	-0.58	-0.48	0.52	-3.32	-1.50	-18.92	10.96	70.38	13.9
3	0	6.16	3.15	1.54	-0.33	-0.30	5.56	3.57	-1.00	1.17	13.91	72.47	17.1
4	0	5.88	3.03	2.48	0.78	0.99	7.46	4.80	3.47	4.70	11.51	79.53	12.8
...	...	...	...	...	...	...	...	...	...	...	...	...	...
31831	0	33.26	24.37	0.00	0.00	0.00	3.23	3.23	-200.00	-200.00	33.33	50.00	50.0
31832	0	33.26	24.37	0.00	0.00	0.00	12.50	12.50	-200.00	-200.00	33.33	50.00	50.0
31833	0	33.26	24.37	0.00	0.00	0.00	11.11	11.11	0.00	0.00	33.33	50.00	50.0
31834	0	33.26	0.00	-33.33	-33.33	-11.11	11.11	0.00	-12.50	-12.50	33.33	50.00	50.0
31835	0	33.26	24.37	0.00	0.00	0.00	0.85	0.85	0.00	0.00	33.33	50.00	50.0

31836 rows × 47 columns

Il dataset di base con il quale si avviano le operazioni è composto da 31836 osservazioni e 47 variabili numeriche (46 features + 1 target) delle quali si presenta una panoramica:

```
credit.dtypes
```

```
FLAG di sana in liquid + anomala        int64
EBITDA/Ricavi                         float64
EBIT/ricavi                           float64
Utile corrente/ricavi                 float64
Risultato netto rettif/ricavi         float64
Risultato netto rettif/AN             float64
EBITDA/AN                             float64
ROA                                   float64
ROE                                   float64
ROE ante imposte                      float64
Val Agg Oper/Ricavi                   float64
Consumi/costi operativi               float64
servizi esterni/costi operativi       float64
costo lavoro/costi operativi          float64
ammortam materiali/costi operativi    float64
Val Agg/ITN                           float64
Ricavi/AN                             float64
gg magazz                             float64
AC/PC                                 float64
AC-mag/PC                             float64
Liq/PC                                float64
Liq/AN                                float64
Cap Circ/AN                           float64
Patr netto/AN                         float64
Riserve+utile/AN                      float64
Patr netto tang/AN                    float64
Patr netto/debiti totali              float64
Patr netto tan/Debiti tot+PN          float64
Patr netto tan/Debiti tot-Liq+PN      float64
OF/RIC                                float64
OFN/RIC                               float64
OFN/EBITDA                            float64
OFN/EBIT                              float64
OFN/AN                                float64
OFN/Autof lordo                       float64
Autof Lordo/AN (=cash flow/attivo)    float64
Auto Lordo-comp straord/AN            float64
Deb totali/VA                         float64
Deb totali/Ric                        float64
PC/ric                                float64
Deb finanziari (stimati)/VA           float64
Deb finanziari (stimati)/Ric          float64
Debiti totali/EBITDA                  float64
Debiti finanziari/EBITDA              float64
Ebitda-servizio debito/AN             float64
Ln(AN)                                float64
Ln(RIC)                               float64
dtype: object
```

Sviluppo dei modelli

Regressione Logistica - Base

(target *anno*)

Inizialmente si importano la maggior parte delle librerie necessarie per utilizzare le funzioni caratteristiche del Machine Learning:

```
from sklearn.datasets import make_classification
from sklearn.metrics import classification_report
from matplotlib import pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import log_loss
import seaborn as sns
```

si va a calcolare il coefficiente di correlazione tra features del dataset e la variabile target:

```
for i in range(len(cols)):
    print(np.corrcoef(credit[cols[i]],credit['FLAG di sana in liquid + anomala'])[0][1], cols[i])
```

```
-0.10218066042112324 EBITDA/Ricavi
-0.11479992897606874 EBIT/ricavi
-0.12193469628262106 Utile corrente/ricavi
-0.11730928946694305 Risultato netto rettif/ricavi
-0.09075043790009757 Risultato netto rettif/AN
-0.0803236419550177 EBITDA/AN
-0.08795206907705984 ROA
-0.14074773977446473 ROE
-0.13666098163815885 ROE ante imposte
-0.08251855065179131 Val Agg Oper/Ricavi
-0.04427025776426584 Consumi/costi operativi
0.053898054891516556 servizi esterni/costi operativi
-0.009852678942679032 costo lavoro/costi operativi
-0.011815561770007109 ammortam materiali/costi operativi
0.004675374015736974 Val Agg/ITN
-0.03447796825438446 Ricavi/AN
-0.014207668390454646 gg magazz
-0.027388854732049977 AC/PC
-0.019763467140784162 AC-mag/PC
-0.006916852586179133 Liq/PC
0.024338153270160492 Liq/AN
-0.07646929568047466 Cap Circ/AN
-0.07834870062925418 Patr netto/AN
```

```
-0.12028736268087954 Riserve+utile/AN
-0.08421474350168877 Patr netto tang/AN
-0.025495450551050104 Patr netto/debiti totali
-0.08045478654252435 Patr netto tan/Debiti tot+PN
-0.06102248731079824 Patr netto tan/Debiti tot-Liq+PN
0.06217098850178526 OF/RIC
0.07001271707153801 OFN/RIC
0.10702192676167478 OFN/EBITDA
0.0921367958535582 OFN/EBIT
0.04094021531213745 OFN/AN
0.09851115296953875 OFN/Autof lordo
-0.0787525892527196 Autof Lordo/AN (=cash flow/attivo)
-0.08927408569813909 Auto Lordo-comp straord/AN
0.042751493575680616 Deb totali/VA
0.07594841599454023 Deb totali/Ric
0.0880111214675612 PC/ric
0.055330667394466584 Deb finanziari (stimati)/VA
0.059470967327155386 Deb finanziari (stimati)/Ric
0.10570246310733099 Debiti totali/EBITDA
0.10803853054092336 Debiti finanziari/EBITDA
-0.09901403113801903 Ebitda-servizio debito/AN
-0.024925050163876774 Ln(AN)
-0.04401872947783763 Ln(RIC)
```

i coefficienti sono rappresentativi dell'importanza cha ha ogni features rispetto alla target. Successivamente, con la funzione *.corr()* si determina, invece, il coefficiente di correlazione fra le features producendo come risposta una matrice nxn, dove n è pari al numero di variabili presenti all'interno del dataset; dunque, nel caso in esame l'output sarà una matrice 47x47; di seguito un estratto:

```
corr = credit.corr()
corr
```

	FLAG di sana in liquid + anomala	EBITDA/Ricavi	EBIT/ricavi	Utile corrente/ricavi	Risultato netto rettif/ricavi	Risultato netto rettif/AN	EBITDA/AN	ROA	ROE	ROE ante imposte
FLAG di sana in liquid + anomala	1.000000	-0.102181	-0.114800	-0.121935	-0.117309	-0.090750	-0.080324	-0.087952	-0.140748	-0.136661
EBITDA/Ricavi	-0.102181	1.000000	0.934246	0.872333	0.789507	0.554818	0.653674	0.614920	0.467868	0.471486
EBIT/ricavi	-0.114800	0.934246	1.000000	0.942071	0.856768	0.564002	0.616030	0.628604	0.500212	0.508534
Utile corrente/ricavi	-0.121935	0.872333	0.942071	1.000000	0.906774	0.590461	0.605429	0.630646	0.527116	0.533833
Risultato netto rettif/ricavi	-0.117309	0.789507	0.856768	0.906774	1.000000	0.657354	0.559377	0.579023	0.554636	0.557535
Risultato netto rettif/AN	-0.090750	0.554818	0.564002	0.590461	0.657354	1.000000	0.809970	0.888596	0.642664	0.650660
EBITDA/AN	-0.080324	0.653674	0.616030	0.605429	0.559377	0.809970	1.000000	0.921851	0.538916	0.572854
ROA	-0.087952	0.614920	0.628604	0.630646	0.579023	0.888596	0.921851	1.000000	0.576080	0.607011
ROE	-0.140748	0.467868	0.500212	0.527116	0.554636	0.642664	0.538916	0.576080	1.000000	0.975761

Quest'ultimi due risultati sono fondamentale per effettuare una prima selezione di features (*features selection*) al fine di decidere quali devono essere rimosse dal dataset; in particolare, dopo aver evidenziato nella matrice tutte le coppie di variabili molto correlate tra loro (correlazione ≥ 0.9) si rimuove quella che registra una minor correlazione (in valore assoluto) con la variabile target e che quindi ha minore importanza. Ad esempio, selezionando le features EBITDA/Ricavi e EBIT/Ricavi, che hanno una correlazione di oltre il 93%, si procederà alla rimozione della variabile EBITDA/Ricavi dato che ha una correlazione con la target di -0.1021 (in valore assoluto < -0.1147 dell'EBIT/Ricavi). Utilizzando un modello di regressione logistica (ma in generale di Machine Learning), questa procedura è ritenuta indispensabile in quanto si preferisce disporre di tutte le variabili più "importanti" e, soprattutto, non altamente correlate tra loro.

Le features da rimuovere risultano le seguenti:

```
elim= ['EBITDA/Ricavi', 'EBIT/ricavi', 'Risultato netto rettif/ricavi',
       'EBITDA/AN', 'ROE ante imposte', 'AC-mag/PC', 'Patr netto/AN',
       'Patr netto tan/Debiti tot+PN', 'Patr netto tan/Debiti tot-Liq+PN',
       'OF/RIC', 'OFN/EBITDA', 'Autof Lordo/AN (=cash flow/attivo)',
       'Deb totali/Ric', 'Debiti totali/EBITDA']
```

```
credit= credit.drop(elim, axis=1)

credit
```

	FLAG di sana in liquid + anomala	Utile corrente/ricavi	Risultato netto rettif/AN	ROA	ROE	Val Agg Oper/Ricavi	Consumi/costi operativi	servizi esterni/costi operativi	costo lavoro/costi operativi
0	0	-8.65	-6.82	-2.65	-19.40	11.48	56.33	19.05	14.76
1	0	1.43	0.64	2.08	1.82	15.32	65.67	15.45	11.52
2	0	-6.01	-0.48	-3.32	-1.50	10.96	70.38	13.95	9.65
3	0	1.54	-0.30	3.57	-1.00	13.91	72.47	17.18	7.71
4	0	2.48	0.99	4.80	3.47	11.51	79.53	12.88	5.64
...	...	...	...	...	...	...	...	...	...
31831	0	0.00	0.00	3.23	-200.00	33.33	50.00	50.00	0.00
31832	0	0.00	0.00	12.50	-200.00	33.33	50.00	50.00	0.00
31833	0	0.00	0.00	11.11	0.00	33.33	50.00	50.00	0.00
31834	0	-33.33	-11.11	0.00	-12.50	33.33	50.00	50.00	0.00
31835	0	0.00	0.00	0.85	0.00	33.33	50.00	50.00	0.00

31836 rows × 33 columns

Eliminando 14 features si arriva ad avere un dataset di 33 variabili (32 features + 1 target).

Si mostrano nuovamente i coefficienti di correlazione delle features rimaste rispetto alla target:

```
for i in range(len(cols)):
    print(np.corrcoef(credit[cols[i]],credit['FLAG di sana in liquid + anomala'])[0][1], cols[i])
```

```
-0.12193469628262106 Utile corrente/ricavi
-0.09075043790009757 Risultato netto rettif/AN
-0.08795206907705984 ROA
-0.14074773977446473 ROE
-0.08251855065179131 Val Agg Oper/Ricavi
-0.04427025776426584 Consumi/costi operativi
0.053898054891516556 servizi esterni/costi operativi
-0.009852678942679032 costo lavoro/costi operativi
-0.011815561770007109 ammortam materiali/costi operativi
0.004675374015736974 Val Agg/ITN
-0.03447796825438446 Ricavi/AN
-0.014207668390454646 gg magazz
-0.027388854732049977 AC/PC
-0.006916852586179133 Liq/PC
0.024338153270160492 Liq/AN
-0.07646929568047466 Cap Circ/AN
-0.12028736268087954 Riserve+utile/AN
-0.08421474350168877 Patr netto tang/AN
-0.025495450551050104 Patr netto/debiti totali
0.07001271707153801 OFN/RIC
0.0921367958535582 OFN/EBIT
0.04094021531213745 OFN/AN
0.09851115296953875 OFN/Autof lordo
-0.08927408569813909 Auto Lordo-comp straord/AN
0.042751493575680616 Deb totali/VA
0.0880111214675612 PC/ric
0.055330667394466584 Deb finanziari (stimati)/VA
0.059470967327155386 Deb finanziari (stimati)/Ric
0.108038530540923 36 Debiti finanziari/EBITDA
-0.09901403113801903 Ebitda-servizio debito/AN
-0.024925050163876774 Ln(AN)
-0.044018729477837 63 Ln(RIC)
```

Il secondo step della features selection consiste nel *rimuovere a priori* quelle variabili che presentano un coefficiente di correlazione con segno "errato". In particolare, tanto più la probabilità in uscita da un modello *logit* è alta, tanto più sarà alta la probabilità di default (probabilità di appartenenza alla classe delle anomale); è per questo che ogni variabile entrante in un modello di regressione logistica deve avere un preciso segno determinato dalla natura

stessa della variabile: se questa ha verso positivo rispetto al fenomeno del default allora tale variabile deve necessariamente entrare nel modello con un coefficiente positivo dato che all'aumentare del suo valore aumenta conseguentemente la probabilità di default; un esempio è rappresentato dalla feature OF/RIC la quale, avendo verso concorde rispetto al default (se aumenta il rapporto tra gli oneri finanziari e il margine operativo lordo teoricamente la società sarà più soggetta al default), dovrà avere coefficiente > 0. Al contrario, una variabile con verso opposto al default (ovvero che all'aumentare della variabile diminuisce la probabilità di default) dovrà avere coefficiente negativo, come accade con l'indice ROE. Nella seguente tabella si riportano i segni con i quali le varie features dovranno entrare nel modello:

X	segno teorico	X	segno teorico
Utile corrente/ricavi	-	Riserve+utile/AN	-
Risultato netto rettif/AN	-	Riserve+utile/AN	-
ROA	-	Patr netto/debiti totali	-
ROE	-	OFN/RIC	+
Val Agg Oper/Ricavi	-	OFN/EBIT	+
Consumi/costi operativi	+	OFN/AN	+
Servizi esterni/costi operativi	+	OFN/Autof lordo	+
costo lavoro/costi operativi	+	Auto Lordo-comp straord/AN	-
ammortam materiali/costi operativi	+	Deb totali/VA	+
Val Agg/ITN	-	PC/ric	+
Ricavi/AN	-	Deb finanziari (stimati)/VA	+
gg magazz	-/+	Deb finanziari (stimati)/Ric	+
AC/PC	-	Debiti finanziari/EBITDA	+
Liq/PC	-	Ebitda-servizio debito/AN	-
Liq/AN	-	Ln(AN)	-
Cap Circ/AN	-	Ln(RIC)	-

Figura 11: Segni teorici corretti

Dunque, dopo aver diviso le features (X_i) e la target (y) in due distinti dataframe, vengono rimosse altre 5 variabili:

```
X = credit.loc[:, credit.columns != 'FLAG di sana in liquid + anomala']
y = credit.loc[:, credit.columns == 'FLAG di sana in liquid + anomala']
```

```
X
```

	Utile corrente/ricavi	Risultato netto rettif/AN	ROA	ROE	Val Agg Oper/Ricavi	Consumi/costi operativi	servizi esterni/costi operativi	costo lavoro/costi operativi	ammortam materiali/costi operativi	Val Agg/ITN
0	-8.65	-6.82	-2.65	-19.40	11.48	56.33	19.05	14.76	9.86	12.78
1	1.43	0.64	2.08	1.82	15.32	65.67	15.45	11.52	7.36	23.70
2	-6.01	-0.48	-3.32	-1.50	10.96	70.38	13.95	9.65	6.02	23.58
3	1.54	-0.30	3.57	-1.00	13.91	72.47	17.18	7.71	2.63	51.95
4	2.48	0.99	4.80	3.47	11.51	79.53	12.88	5.64	1.94	67.62
...	...	...	...	...	...	...	...	...	...	...
31831	0.00	0.00	3.23	-200.00	33.33	50.00	50.00	0.00	0.00	4.55
31832	0.00	0.00	12.50	-200.00	33.33	50.00	50.00	0.00	0.00	100.00
31833	0.00	0.00	11.11	0.00	33.33	50.00	50.00	0.00	0.00	100.00
31834	-33.33	-11.11	0.00	-12.50	33.33	50.00	50.00	0.00	0.00	100.00
31835	0.00	0.00	0.85	0.00	33.33	50.00	50.00	0.00	0.00	100.00

31836 rows × 32 columns

```
y #y_anno
```

	FLAG di sana in liquid + anomala
0	0
1	0
2	0
3	0
4	0
...	...
31831	0
31832	0
31833	0
31834	0
31835	0

31836 rows × 1 columns

```
elim= ['Consumi/costi operativi','costo lavoro/costi operativi',
       'ammortam materiali/costi operativi','Val Agg/ITN','Liq/AN']

X= X.drop(elim, axis=1)
X
```

	Utile corrente/ricavi	Risultato netto rettif/AN	ROA	ROE	Val Agg Oper/Ricavi	servizi esterni/costi operativi	Ricavi/AN	gg magazz	AC/PC	Liq/PC
0	-8.65	-6.82	-2.65	-19.40	11.48	19.05	0.52	161.21	105.06	1.79
1	1.43	0.64	2.08	1.82	15.32	15.45	0.68	140.68	138.75	3.36
2	-6.01	-0.48	-3.32	-1.50	10.96	13.95	0.82	116.49	129.64	1.60
3	1.54	-0.30	3.57	-1.00	13.91	17.18	0.90	154.54	142.72	6.61
4	2.48	0.99	4.80	3.47	11.51	12.88	1.27	125.27	130.84	4.14
...	...	...	...	...	...	...	...	...	...	...
31831	0.00	0.00	3.23	-200.00	33.33	50.00	0.10	0.00	35.23	6.15
31832	0.00	0.00	12.50	-200.00	33.33	50.00	0.38	226.23	60.00	10.00
31833	0.00	0.00	11.11	0.00	33.33	50.00	0.33	0.00	600.00	231.28
31834	-33.33	-11.11	0.00	-12.50	33.33	50.00	0.33	0.00	600.00	231.28
31835	0.00	0.00	0.85	0.00	33.33	50.00	0.03	0.00	600.00	231.28

31836 rows × 27 columns

Si arriva dunque a un dataset composto da 27 features. Da sottolineare è il caso della rimozione della variabile *Liq/AN*, la quale deve entrare nel modello con un coefficiente negativo ma presenta una correlazione con la target positiva. A questo fatto si può dare la seguente interpretazione: dato che la target *anno* (e anche quella *società*) è stata costruita sia con i casi di default sia con le liquidazioni, tutte le società liquidate che nell'ultima fase del processo di liquidazione hanno cominciato a vendere gli asset registreranno un tasso di liquidità sull'attivo più alto del normale (dovuto alla svendita); dunque, risulta chiaro che nel campione è presente un certo numero di società in questa situazione che inevitabilmente va a sporcare la variabile *Liq/AN*.

Adesso che ogni variabile ha la "corretta" correlazione con la target, si va a controllare il segno del coefficiente con il quale le varie features effettivamente entrano nel modello:

```
import statsmodels.api as sm
logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
                          Results: Logit
==========================================================================
Model:              Logit                          Pseudo R-squared: 0.173
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1719.2482
Date:               2020-11-16 12:13               BIC:              1945.1937
No. Observations:   31836                          Log-Likelihood:   -832.62
Df Model:           26                             LL-Null:          -1006.5
Df Residuals:       31809                          LLR p-value:      5.2198e-58
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
--------------------------------------------------------------------------
                                  Coef.  Std.Err.    z    P>|z|   [0.025  0.975]
--------------------------------------------------------------------------
Utile corrente/ricavi            -0.0070   0.0040 -1.7450 0.0810 -0.0149  0.0009
Risultato netto rettif/AN         0.0031   0.0202  0.1516 0.8795 -0.0366  0.0427
ROA                               0.0337   0.0275  1.2274 0.2197 -0.0201  0.0876
ROE                              -0.0047   0.0017 -2.7846 0.0054 -0.0080 -0.0014
Val Agg Oper/Ricavi              -0.0122   0.0041 -2.9966 0.0027 -0.0202 -0.0042
servizi esterni/costi operativi  -0.0205   0.0038 -5.3621 0.0000 -0.0280 -0.0130
Ricavi/AN                        -1.8119   0.2204 -8.2214 0.0000 -2.2439 -1.3800
gg magazz                        -0.0038   0.0013 -2.8494 0.0044 -0.0065 -0.0012
AC/PC                            -0.0065   0.0018 -3.6336 0.0003 -0.0099 -0.0030
Liq/PC                            0.0064   0.0026  2.4469 0.0144  0.0013  0.0115
Cap Circ/AN                       0.0093   0.0051  1.8208 0.0686 -0.0007  0.0193
Riserve+utile/AN                 -0.0244   0.0046 -5.2574 0.0000 -0.0334 -0.0153
Patr netto tang/AN               -0.0129   0.0066 -1.9620 0.0498 -0.0258 -0.0000
Patr netto/debiti totali          0.0040   0.0013  3.0284 0.0025  0.0014  0.0066
OFN/RIC                           0.0271   0.0289  0.9384 0.3481 -0.0295  0.0837
OFN/EBIT                          0.0008   0.0010  0.7811 0.4347 -0.0012  0.0028
OFN/AN                           -0.1322   0.0940 -1.4056 0.1598 -0.3165  0.0521
OFN/Autof lordo                  -0.0008   0.0010 -0.7682 0.4424 -0.0027  0.0012
Auto Lordo-comp straord/AN       -0.0535   0.0299 -1.7915 0.0732 -0.1120  0.0050
Deb totali/VA                    -0.0003   0.0003 -1.0047 0.3151 -0.0008  0.0003
PC/ric                           -0.0001   0.0003 -0.2622 0.7932 -0.0007  0.0005
Deb finanziari (stimati)/VA       0.0012   0.0005  2.7141 0.0066  0.0003  0.0021
Deb finanziari (stimati)/Ric     -0.0001   0.0003 -0.2987 0.7652 -0.0008  0.0006
Debiti finanziari/EBITDA          0.0006   0.0004  1.5871 0.1125 -0.0002  0.0014
Ebitda-servizio debito/AN         0.0386   0.0175  2.2034 0.0276  0.0043  0.0730
Ln(AN)                           -1.0853   0.1141 -9.5106 0.0000 -1.3090 -0.8616
Ln(RIC)                           0.8145   0.1144  7.1179 0.0000  0.5903  1.0388
==========================================================================
```

A questo punto si procede con l'ultima fase di features selection: attraverso un processo iterativo, ad ogni step, si andrà a rimuovere dal modello la feature (una alla volta) che ha il coefficiente con il segno economico-teorico errato (dalla tabella) e che registra il p-value maggiore. I risultati indicano che la prima feature da rimuovere è *Risultato netto rettif/AN* perché tra quelle con segno errato (entra positiva quando deve essere negativa) è quella con p-value maggiore (87.95%); dopo aver rimosso la variabile si stima nuovamente il modello e si effettua un altro step del processo:

```
X= X.drop(['Risultato netto rettif/AN'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
                          Results: Logit
==========================================================================
Model:              Logit                            Pseudo R-squared: 0.173
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1717.2712
Date:               2020-11-16 12:27                 BIC:              1934.8483
No. Observations:   31836                            Log-Likelihood:   -832.64
Df Model:           25                               LL-Null:          -1006.5
Df Residuals:       31810                            LLR p-value:      1.3963e-58
Converged:          1.0000                           Scale:            1.0000
No. Iterations:     10.0000
--------------------------------------------------------------------------
                                  Coef.  Std.Err.    z    P>|z|   [0.025  0.975]
--------------------------------------------------------------------------
Utile corrente/ricavi            -0.0069   0.0040 -1.7398 0.0819 -0.0148  0.0009
ROA                               0.0357   0.0243  1.4675 0.1422 -0.0120  0.0833
ROE                              -0.0047   0.0017 -2.7787 0.0055 -0.0080 -0.0014
Val Agg Oper/Ricavi              -0.0123   0.0041 -3.0197 0.0025 -0.0202 -0.0043
servizi esterni/costi operativi  -0.0205   0.0038 -5.3610 0.0000 -0.0280 -0.0130
Ricavi/AN                        -1.8120   0.2203 -8.2266 0.0000 -2.2437 -1.3803
gg magazz                        -0.0038   0.0013 -2.8493 0.0044 -0.0065 -0.0012
AC/PC                            -0.0064   0.0018 -3.6308 0.0003 -0.0099 -0.0030
Liq/PC                            0.0064   0.0026  2.4441 0.0145  0.0013  0.0114
Cap Circ/AN                       0.0093   0.0051  1.8217 0.0685 -0.0007  0.0193
Riserve+utile/AN                 -0.0244   0.0046 -5.2589 0.0000 -0.0335 -0.0153
Patr netto tang/AN               -0.0129   0.0066 -1.9670 0.0492 -0.0258 -0.0000
Patr netto/debiti totali          0.0040   0.0013  3.0269 0.0025  0.0014  0.0066
OFN/RIC                           0.0273   0.0288  0.9467 0.3438 -0.0292  0.0839
OFN/EBIT                          0.0008   0.0010  0.8028 0.4221 -0.0012  0.0028
OFN/AN                           -0.1332   0.0938 -1.4196 0.1557 -0.3171  0.0507
OFN/Autof lordo                  -0.0009   0.0008 -1.1598 0.2461 -0.0024  0.0006
Auto Lordo-comp straord/AN       -0.0537   0.0298 -1.8040 0.0712 -0.1120  0.0046
Deb totali/VA                    -0.0003   0.0003 -1.0029 0.3159 -0.0008  0.0003
PC/ric                           -0.0001   0.0003 -0.2529 0.8003 -0.0007  0.0005
Deb finanziari (stimati)/VA       0.0012   0.0005  2.7311 0.0063  0.0003  0.0021
Deb finanziari (stimati)/Ric     -0.0001   0.0003 -0.2991 0.7649 -0.0008  0.0006
Debiti finanziari/EBITDA          0.0007   0.0004  1.7022 0.0887 -0.0001  0.0014
Ebitda-servizio debito/AN         0.0388   0.0175  2.2164 0.0267  0.0045  0.0731
Ln(AN)                           -1.0848   0.1140 -9.5120 0.0000 -1.3084 -0.8613
Ln(RIC)                           0.8142   0.1144  7.1188 0.0000  0.5900  1.0383
==========================================================================
```

```
X= X.drop(['PC/ric'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
=========================================================================================
Model:              Logit                            Pseudo R-squared: 0.173
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1715.3351
Date:               2020-11-16 12:31                 BIC:              1924.5439
No. Observations:   31836                            Log-Likelihood:   -832.67
Df Model:           24                               LL-Null:          -1006.5
Df Residuals:       31811                            LLR p-value:      3.7291e-59
Converged:          1.0000                           Scale:            1.0000
No. Iterations:     10.0000
-----------------------------------------------------------------------------------------
                                  Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
-----------------------------------------------------------------------------------------
Utile corrente/ricavi            -0.0068   0.0040  -1.7234 0.0848 -0.0146  0.0009
ROA                               0.0353   0.0243   1.4552 0.1456 -0.0123  0.0829
ROE                              -0.0046   0.0017  -2.7628 0.0057 -0.0079 -0.0013
Val Agg Oper/Ricavi              -0.0122   0.0040  -3.0092 0.0026 -0.0201 -0.0042
servizi esterni/costi operativi  -0.0207   0.0038  -5.4828 0.0000 -0.0281 -0.0133
Ricavi/AN                        -1.8155   0.2201  -8.2489 0.0000 -2.2468 -1.3841
gg magazz                        -0.0038   0.0013  -2.8581 0.0043 -0.0065 -0.0012
AC/PC                            -0.0065   0.0018  -3.6258 0.0003 -0.0100 -0.0030
Liq/PC                            0.0064   0.0026   2.4633 0.0138  0.0013  0.0115
Cap Circ/AN                       0.0096   0.0050   1.9250 0.0542 -0.0002  0.0194
Riserve+utile/AN                 -0.0243   0.0046  -5.2542 0.0000 -0.0334 -0.0153
Patr netto tang/AN               -0.0129   0.0066  -1.9655 0.0494 -0.0258 -0.0000
Patr netto/debiti totali          0.0040   0.0013   3.0586 0.0022  0.0014  0.0066
OFN/RIC                           0.0267   0.0287   0.9294 0.3527 -0.0296  0.0830
OFN/EBIT                          0.0008   0.0010   0.8009 0.4232 -0.0012  0.0028
OFN/AN                           -0.1291   0.0924  -1.3974 0.1623 -0.3103  0.0520
OFN/Autof lordo                  -0.0009   0.0008  -1.1458 0.2519 -0.0023  0.0006
Auto Lordo-comp straord/AN       -0.0535   0.0298  -1.7971 0.0723 -0.1118  0.0048
Deb totali/VA                    -0.0003   0.0003  -1.0137 0.3107 -0.0008  0.0003
Deb finanziari (stimati)/VA       0.0012   0.0005   2.7306 0.0063  0.0003  0.0021
Deb finanziari (stimati)/Ric     -0.0001   0.0003  -0.3973 0.6911 -0.0008  0.0005
Debiti finanziari/EBITDA          0.0007   0.0004   1.6943 0.0902 -0.0001  0.0014
Ebitda-servizio debito/AN         0.0382   0.0173   2.2042 0.0275  0.0042  0.0722
Ln(AN)                           -1.0941   0.1076 -10.1644 0.0000 -1.3051 -0.8832
Ln(RIC)                           0.8231   0.1085   7.5881 0.0000  0.6105  1.0357
=========================================================================================
```

```
X= X.drop(['Deb finanziari (stimati)/Ric'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
================================================================================
Model:              Logit                          Pseudo R-squared: 0.173
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1713.4932
Date:               2020-11-16 12:33               BIC:              1914.3337
No. Observations:   31836                          Log-Likelihood:   -832.75
Df Model:           23                             LL-Null:          -1006.5
Df Residuals:       31812                          LLR p-value:      1.0188e-59
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
--------------------------------------------------------------------------------
                                  Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
--------------------------------------------------------------------------------
Utile corrente/ricavi            -0.0068   0.0040  -1.7151 0.0863 -0.0145  0.0010
ROA                               0.0335   0.0238   1.4074 0.1593 -0.0131  0.0801
ROE                              -0.0046   0.0017  -2.7618 0.0057 -0.0079 -0.0013
Val Agg Oper/Ricavi              -0.0121   0.0040  -2.9940 0.0028 -0.0200 -0.0042
servizi esterni/costi operativi  -0.0207   0.0038  -5.4832 0.0000 -0.0280 -0.0133
Ricavi/AN                        -1.8223   0.2195  -8.3029 0.0000 -2.2524 -1.3921
gg magazz                        -0.0038   0.0013  -2.8423 0.0045 -0.0064 -0.0012
AC/PC                            -0.0065   0.0018  -3.6198 0.0003 -0.0099 -0.0030
Liq/PC                            0.0064   0.0026   2.4578 0.0140  0.0013  0.0115
Cap Circ/AN                       0.0095   0.0050   1.9155 0.0554 -0.0002  0.0193
Riserve+utile/AN                 -0.0244   0.0046  -5.2781 0.0000 -0.0335 -0.0154
Patr netto tang/AN               -0.0128   0.0066  -1.9477 0.0514 -0.0257  0.0001
Patr netto/debiti totali          0.0040   0.0013   3.0735 0.0021  0.0015  0.0066
OFN/RIC                           0.0220   0.0261   0.8417 0.4000 -0.0292  0.0731
OFN/EBIT                          0.0008   0.0010   0.7843 0.4329 -0.0012  0.0028
OFN/AN                           -0.1200   0.0894  -1.3423 0.1795 -0.2952  0.0552
OFN/Autof lordo                  -0.0008   0.0007  -1.1167 0.2641 -0.0023  0.0006
Auto Lordo-comp straord/AN       -0.0556   0.0294  -1.8941 0.0582 -0.1131  0.0019
Deb totali/VA                    -0.0003   0.0003  -0.9977 0.3184 -0.0008  0.0003
Deb finanziari (stimati)/VA       0.0012   0.0005   2.7138 0.0067  0.0003  0.0021
Debiti finanziari/EBITDA          0.0007   0.0004   1.6953 0.0900 -0.0001  0.0014
Ebitda-servizio debito/AN         0.0412   0.0157   2.6235 0.0087  0.0104  0.0720
Ln(AN)                           -1.1072   0.1024 -10.8149 0.0000 -1.3079 -0.9065
Ln(RIC)                           0.8367   0.1029   8.1331 0.0000  0.6351  1.0384
================================================================================
```

```
X= X.drop(['Deb totali/VA'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
================================================================================
Model:              Logit                          Pseudo R-squared: 0.172
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1712.5170
Date:               2020-11-16 12:34               BIC:              1904.9891
No. Observations:   31836                          Log-Likelihood:   -833.26
Df Model:           22                             LL-Null:          -1006.5
Df Residuals:       31813                          LLR p-value:      4.0941e-60
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
--------------------------------------------------------------------------------
                                 Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
--------------------------------------------------------------------------------
Utile corrente/ricavi           -0.0070   0.0039  -1.7726 0.0763 -0.0147  0.0007
ROA                              0.0352   0.0238   1.4797 0.1389 -0.0114  0.0817
ROE                             -0.0046   0.0017  -2.7253 0.0064 -0.0078 -0.0013
Val Agg Oper/Ricavi             -0.0130   0.0039  -3.3046 0.0010 -0.0207 -0.0053
servizi esterni/costi operativi -0.0206   0.0038  -5.4796 0.0000 -0.0280 -0.0132
Ricavi/AN                       -1.8389   0.2190  -8.3980 0.0000 -2.2681 -1.4097
gg magazz                       -0.0039   0.0013  -2.8984 0.0038 -0.0065 -0.0013
AC/PC                           -0.0066   0.0018  -3.6779 0.0002 -0.0101 -0.0031
Liq/PC                           0.0064   0.0026   2.4777 0.0132  0.0013  0.0115
Cap Circ/AN                      0.0099   0.0050   2.0069 0.0448  0.0002  0.0197
Riserve+utile/AN                -0.0243   0.0046  -5.2552 0.0000 -0.0333 -0.0152
Patr netto tang/AN              -0.0128   0.0066  -1.9504 0.0511 -0.0257  0.0001
Patr netto/debiti totali         0.0042   0.0013   3.2245 0.0013  0.0016  0.0067
OFN/RIC                          0.0215   0.0262   0.8203 0.4120 -0.0298  0.0728
OFN/EBIT                         0.0008   0.0010   0.7609 0.4467 -0.0012  0.0028
OFN/AN                          -0.1109   0.0890  -1.2461 0.2127 -0.2852  0.0635
OFN/Autof lordo                 -0.0008   0.0007  -1.0982 0.2721 -0.0023  0.0006
Auto Lordo-comp straord/AN      -0.0503   0.0288  -1.7446 0.0811 -0.1068  0.0062
Deb finanziari (stimati)/VA      0.0009   0.0003   3.0582 0.0022  0.0003  0.0014
Debiti finanziari/EBITDA         0.0007   0.0004   1.7461 0.0808 -0.0001  0.0014
Ebitda-servizio debito/AN        0.0369   0.0151   2.4442 0.0145  0.0073  0.0666
Ln(AN)                          -1.1196   0.1016 -11.0178 0.0000 -1.3187 -0.9204
Ln(RIC)                          0.8441   0.1026   8.2257 0.0000  0.6430  1.0452
================================================================================
```

```
X= X.drop(['OFN/Autof lordo'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
================================================================================
Model:              Logit                          Pseudo R-squared: 0.172
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1711.6913
Date:               2020-11-16 12:35               BIC:              1895.7951
No. Observations:   31836                          Log-Likelihood:   -833.85
Df Model:           21                             LL-Null:          -1006.5
Df Residuals:       31814                          LLR p-value:      1.7299e-60
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
--------------------------------------------------------------------------------
                                 Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
--------------------------------------------------------------------------------
Utile corrente/ricavi           -0.0068   0.0039  -1.7379 0.0822 -0.0146  0.0009
ROA                              0.0320   0.0235   1.3645 0.1724 -0.0140  0.0781
ROE                             -0.0042   0.0017  -2.5612 0.0104 -0.0075 -0.0010
Val Agg Oper/Ricavi             -0.0131   0.0039  -3.3465 0.0008 -0.0208 -0.0054
servizi esterni/costi operativi -0.0205   0.0038  -5.4551 0.0000 -0.0279 -0.0132
Ricavi/AN                       -1.8610   0.2181  -8.5311 0.0000 -2.2885 -1.4334
gg magazz                       -0.0039   0.0013  -2.9461 0.0032 -0.0066 -0.0013
AC/PC                           -0.0066   0.0018  -3.6991 0.0002 -0.0101 -0.0031
Liq/PC                           0.0065   0.0026   2.4938 0.0126  0.0014  0.0115
Cap Circ/AN                      0.0099   0.0049   2.0037 0.0451  0.0002  0.0196
Riserve+utile/AN                -0.0242   0.0046  -5.2384 0.0000 -0.0332 -0.0151
Patr netto tang/AN              -0.0133   0.0065  -2.0365 0.0417 -0.0261 -0.0005
Patr netto/debiti totali         0.0042   0.0013   3.2326 0.0012  0.0016  0.0067
OFN/RIC                          0.0194   0.0260   0.7457 0.4558 -0.0316  0.0704
OFN/EBIT                         0.0006   0.0010   0.5814 0.5610 -0.0014  0.0026
OFN/AN                          -0.1190   0.0886  -1.3435 0.1791 -0.2926  0.0546
Auto Lordo-comp straord/AN      -0.0430   0.0275  -1.5598 0.1188 -0.0969  0.0110
Deb finanziari (stimati)/VA      0.0008   0.0003   2.9272 0.0034  0.0003  0.0014
Debiti finanziari/EBITDA         0.0006   0.0004   1.5081 0.1315 -0.0002  0.0013
Ebitda-servizio debito/AN        0.0361   0.0150   2.3983 0.0165  0.0066  0.0656
Ln(AN)                          -1.1228   0.1016 -11.0511 0.0000 -1.3219 -0.9237
Ln(RIC)                          0.8484   0.1026   8.2667 0.0000  0.6472  1.0495
================================================================================
```

```
X= X.drop(['OFN/AN'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
================================================================================
Model:              Logit                           Pseudo R-squared: 0.171
Dependent Variable: FLAG di sana in liquid + anomala AIC:             1711.4931
Date:               2020-11-16 12:36                BIC:              1887.2285
No. Observations:   31836                           Log-Likelihood:   -834.75
Df Model:           20                              LL-Null:          -1006.5
Df Residuals:       31815                           LLR p-value:      9.6293e-61
Converged:          1.0000                          Scale:            1.0000
No. Iterations:     10.0000
--------------------------------------------------------------------------------
                                 Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
--------------------------------------------------------------------------------
Utile corrente/ricavi           -0.0071   0.0040  -1.7905 0.0734 -0.0148  0.0007
ROA                              0.0292   0.0234   1.2494 0.2115 -0.0166  0.0750
ROE                             -0.0043   0.0016  -2.5774 0.0100 -0.0075 -0.0010
Val Agg Oper/Ricavi             -0.0133   0.0040  -3.3763 0.0007 -0.0211 -0.0056
servizi esterni/costi operativi -0.0208   0.0038  -5.5274 0.0000 -0.0282 -0.0134
Ricavi/AN                       -1.9239   0.2157  -8.9211 0.0000 -2.3466 -1.5012
gg magazz                       -0.0041   0.0013  -3.0854 0.0020 -0.0068 -0.0015
AC/PC                           -0.0064   0.0018  -3.6369 0.0003 -0.0099 -0.0030
Liq/PC                           0.0064   0.0026   2.4721 0.0134  0.0013  0.0114
Cap Circ/AN                      0.0098   0.0049   1.9890 0.0467  0.0001  0.0195
Riserve+utile/AN                -0.0239   0.0046  -5.1892 0.0000 -0.0330 -0.0149
Patr netto tang/AN              -0.0130   0.0065  -2.0064 0.0448 -0.0258 -0.0003
Patr netto/debiti totali         0.0041   0.0013   3.1820 0.0015  0.0016  0.0066
OFN/RIC                         -0.0059   0.0182  -0.3224 0.7472 -0.0416  0.0299
OFN/EBIT                         0.0005   0.0010   0.4798 0.6314 -0.0015  0.0025
Auto Lordo-comp straord/AN      -0.0422   0.0276  -1.5278 0.1266 -0.0962  0.0119
Deb finanziari (stimati)/VA      0.0008   0.0003   2.9491 0.0032  0.0003  0.0014
Debiti finanziari/EBITDA         0.0006   0.0004   1.6231 0.1046 -0.0001  0.0013
Ebitda-servizio debito/AN        0.0410   0.0147   2.7895 0.0053  0.0122  0.0699
Ln(AN)                          -1.0847   0.0982 -11.0405 0.0000 -1.2773 -0.8922
Ln(RIC)                          0.8065   0.0981   8.2206 0.0000  0.6142  0.9988
================================================================================
```

```
X= X.drop(['OFN/RIC'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
================================================================================
Model:              Logit                           Pseudo R-squared: 0.171
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1709.5969
Date:               2020-11-16 12:37                BIC:              1876.9640
No. Observations:   31836                           Log-Likelihood:   -834.80
Df Model:           19                              LL-Null:          -1006.5
Df Residuals:       31816                           LLR p-value:      2.3410e-61
Converged:          1.0000                          Scale:            1.0000
No. Iterations:     10.0000
--------------------------------------------------------------------------------
                                 Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
--------------------------------------------------------------------------------
Utile corrente/ricavi           -0.0068   0.0039  -1.7593 0.0785 -0.0143  0.0008
ROA                              0.0273   0.0226   1.2088 0.2267 -0.0170  0.0715
ROE                             -0.0042   0.0016  -2.5691 0.0102 -0.0075 -0.0010
Val Agg Oper/Ricavi             -0.0135   0.0039  -3.4227 0.0006 -0.0212 -0.0058
servizi esterni/costi operativi -0.0208   0.0038  -5.5335 0.0000 -0.0282 -0.0135
Ricavi/AN                       -1.9278   0.2156  -8.9428 0.0000 -2.3503 -1.5053
gg magazz                       -0.0041   0.0013  -3.1027 0.0019 -0.0068 -0.0015
AC/PC                           -0.0065   0.0018  -3.6466 0.0003 -0.0099 -0.0030
Liq/PC                           0.0064   0.0026   2.4875 0.0129  0.0014  0.0115
Cap Circ/AN                      0.0099   0.0049   2.0105 0.0444  0.0002  0.0195
Riserve+utile/AN                -0.0239   0.0046  -5.1914 0.0000 -0.0330 -0.0149
Patr netto tang/AN              -0.0131   0.0065  -2.0163 0.0438 -0.0258 -0.0004
Patr netto/debiti totali         0.0041   0.0013   3.1841 0.0015  0.0016  0.0066
OFN/EBIT                         0.0005   0.0010   0.4549 0.6492 -0.0015  0.0025
Auto Lordo-comp straord/AN      -0.0413   0.0274  -1.5054 0.1322 -0.0951  0.0125
Deb finanziari (stimati)/VA      0.0008   0.0003   2.9286 0.0034  0.0003  0.0014
Debiti finanziari/EBITDA         0.0006   0.0004   1.6634 0.0962 -0.0001  0.0014
Ebitda-servizio debito/AN        0.0419   0.0145   2.8901 0.0039  0.0135  0.0704
Ln(AN)                          -1.0895   0.0971 -11.2158 0.0000 -1.2799 -0.8991
Ln(RIC)                          0.8123   0.0965   8.4182 0.0000  0.6232  1.0014
================================================================================
```

```
X= X.drop(['ROA'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
=================================================================================
Model:              Logit                          Pseudo R-squared: 0.170
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1709.0471
Date:               2020-11-16 12:38               BIC:              1868.0458
No. Observations:   31836                          Log-Likelihood:   -835.52
Df Model:           18                             LL-Null:          -1006.5
Df Residuals:       31817                          LLR p-value:      1.0521e-61
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
---------------------------------------------------------------------------------
                                 Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------------------
Utile corrente/ricavi           -0.0051   0.0036  -1.4213 0.1552 -0.0122  0.0019
ROE                             -0.0041   0.0016  -2.4953 0.0126 -0.0073 -0.0009
Val Agg Oper/Ricavi             -0.0140   0.0039  -3.5568 0.0004 -0.0217 -0.0063
servizi esterni/costi operativi -0.0203   0.0037  -5.4420 0.0000 -0.0276 -0.0130
Ricavi/AN                       -1.8774   0.2111  -8.8938 0.0000 -2.2911 -1.4637
gg magazz                       -0.0041   0.0013  -3.0562 0.0022 -0.0067 -0.0015
AC/PC                           -0.0064   0.0018  -3.6507 0.0003 -0.0099 -0.0030
Liq/PC                           0.0064   0.0026   2.4763 0.0133  0.0013  0.0115
Cap Circ/AN                      0.0102   0.0049   2.0690 0.0385  0.0005  0.0198
Riserve+utile/AN                -0.0240   0.0046  -5.1970 0.0000 -0.0330 -0.0149
Patr netto tang/AN              -0.0134   0.0065  -2.0788 0.0376 -0.0261 -0.0008
Patr netto/debiti totali         0.0041   0.0013   3.2002 0.0014  0.0016  0.0066
OFN/EBIT                         0.0001   0.0010   0.1243 0.9011 -0.0018  0.0020
Auto Lordo-comp straord/AN      -0.0266   0.0239  -1.1149 0.2649 -0.0734  0.0202
Deb finanziari (stimati)/VA      0.0009   0.0003   3.1514 0.0016  0.0003  0.0014
Debiti finanziari/EBITDA         0.0006   0.0004   1.6959 0.0899 -0.0001  0.0014
Ebitda-servizio debito/AN        0.0443   0.0144   3.0849 0.0020  0.0162  0.0725
Ln(AN)                          -1.0560   0.0928 -11.3746 0.0000 -1.2380 -0.8741
Ln(RIC)                          0.7771   0.0918   8.4632 0.0000  0.5971  0.9571
=================================================================================
```

```
X= X.drop(['Cap Circ/AN'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
=================================================================================
Model:              Logit                          Pseudo R-squared: 0.168
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1711.2347
Date:               2020-11-16 12:39               BIC:              1861.8650
No. Observations:   31836                          Log-Likelihood:   -837.62
Df Model:           17                             LL-Null:          -1006.5
Df Residuals:       31818                          LLR p-value:      1.7062e-61
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
---------------------------------------------------------------------------------
                                 Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------------------
Utile corrente/ricavi           -0.0045   0.0036  -1.2468 0.2125 -0.0115  0.0026
ROE                             -0.0038   0.0016  -2.2964 0.0217 -0.0070 -0.0006
Val Agg Oper/Ricavi             -0.0142   0.0039  -3.5892 0.0003 -0.0219 -0.0064
servizi esterni/costi operativi -0.0211   0.0037  -5.7086 0.0000 -0.0283 -0.0138
Ricavi/AN                       -1.9115   0.2112  -9.0521 0.0000 -2.3254 -1.4976
gg magazz                       -0.0040   0.0013  -2.9891 0.0028 -0.0066 -0.0014
AC/PC                           -0.0040   0.0013  -3.1114 0.0019 -0.0066 -0.0015
Liq/PC                           0.0062   0.0027   2.3111 0.0208  0.0009  0.0114
Riserve+utile/AN                -0.0233   0.0046  -5.0803 0.0000 -0.0323 -0.0143
Patr netto tang/AN              -0.0083   0.0061  -1.3618 0.1733 -0.0203  0.0036
Patr netto/debiti totali         0.0029   0.0012   2.4304 0.0151  0.0006  0.0052
OFN/EBIT                         0.0001   0.0010   0.0974 0.9224 -0.0018  0.0020
Auto Lordo-comp straord/AN      -0.0292   0.0238  -1.2278 0.2195 -0.0759  0.0174
Deb finanziari (stimati)/VA      0.0008   0.0003   3.0076 0.0026  0.0003  0.0014
Debiti finanziari/EBITDA         0.0006   0.0004   1.6991 0.0893 -0.0001  0.0014
Ebitda-servizio debito/AN        0.0453   0.0144   3.1492 0.0016  0.0171  0.0735
Ln(AN)                          -1.0909   0.0913 -11.9548 0.0000 -1.2698 -0.9121
Ln(RIC)                          0.7877   0.0917   8.5888 0.0000  0.6079  0.9674
=================================================================================
```

```
X= X.drop(['Liq/PC'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
=================================================================================
Model:              Logit                          Pseudo R-squared: 0.165
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1714.5570
Date:               2020-11-16 12:41               BIC:              1856.8190
No. Observations:   31836                          Log-Likelihood:   -840.28
Df Model:           16                             LL-Null:          -1006.5
Df Residuals:       31819                          LLR p-value:      4.6709e-61
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
---------------------------------------------------------------------------------
                                 Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------------------
Utile corrente/ricavi           -0.0043   0.0036  -1.1900 0.2340 -0.0114  0.0028
ROE                             -0.0040   0.0017  -2.4027 0.0163 -0.0072 -0.0007
Val Agg Oper/Ricavi             -0.0147   0.0040  -3.7076 0.0002 -0.0224 -0.0069
servizi esterni/costi operativi -0.0213   0.0037  -5.7464 0.0000 -0.0285 -0.0140
Ricavi/AN                       -1.9338   0.2126  -9.0944 0.0000 -2.3505 -1.5170
gg magazz                       -0.0045   0.0013  -3.4425 0.0006 -0.0071 -0.0020
AC/PC                           -0.0022   0.0009  -2.2993 0.0215 -0.0040 -0.0003
Riserve+utile/AN                -0.0240   0.0046  -5.2023 0.0000 -0.0330 -0.0149
Patr netto tang/AN              -0.0082   0.0061  -1.3350 0.1819 -0.0202  0.0038
Patr netto/debiti totali         0.0033   0.0012   2.8086 0.0050  0.0010  0.0056
OFN/EBIT                        -0.0000   0.0010  -0.0074 0.9941 -0.0019  0.0019
Auto Lordo-comp straord/AN      -0.0279   0.0238  -1.1746 0.2402 -0.0745  0.0187
Deb finanziari (stimati)/VA      0.0009   0.0003   3.0330 0.0024  0.0003  0.0014
Debiti finanziari/EBITDA         0.0007   0.0004   1.7338 0.0830 -0.0001  0.0014
Ebitda-servizio debito/AN        0.0456   0.0144   3.1655 0.0015  0.0174  0.0738
Ln(AN)                          -1.0954   0.0917 -11.9406 0.0000 -1.2752 -0.9156
Ln(RIC)                          0.7856   0.0924   8.5015 0.0000  0.6045  0.9667
=================================================================================
```

```
X= X.drop(['OFN/EBIT'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
================================================================================
Model:              Logit                          Pseudo R-squared: 0.165
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1712.5571
Date:               2020-11-16 12:42               BIC:              1846.4507
No. Observations:   31836                          Log-Likelihood:   -840.28
Df Model:           15                             LL-Null:          -1006.5
Df Residuals:       31820                          LLR p-value:      9.7276e-62
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
--------------------------------------------------------------------------------
                                 Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
--------------------------------------------------------------------------------
Utile corrente/ricavi           -0.0043   0.0035  -1.2260 0.2202 -0.0112  0.0026
ROE                             -0.0040   0.0016  -2.4578 0.0140 -0.0071 -0.0008
Val Agg Oper/Ricavi             -0.0147   0.0039  -3.7766 0.0002 -0.0223 -0.0071
servizi esterni/costi operativi -0.0213   0.0037  -5.7550 0.0000 -0.0285 -0.0140
Ricavi/AN                       -1.9339   0.2122  -9.1134 0.0000 -2.3498 -1.5180
gg magazz                       -0.0045   0.0013  -3.4500 0.0006 -0.0071 -0.0020
AC/PC                           -0.0022   0.0009  -2.3004 0.0214 -0.0040 -0.0003
Riserve+utile/AN                -0.0240   0.0046  -5.2035 0.0000 -0.0330 -0.0149
Patr netto tang/AN              -0.0082   0.0061  -1.3438 0.1790 -0.0202  0.0038
Patr netto/debiti totali         0.0033   0.0012   2.8088 0.0050  0.0010  0.0056
Auto Lordo-comp straord/AN      -0.0279   0.0237  -1.1788 0.2385 -0.0743  0.0185
Deb finanziari (stimati)/VA      0.0009   0.0003   3.0667 0.0022  0.0003  0.0014
Debiti finanziari/EBITDA         0.0006   0.0003   2.0590 0.0395  0.0000  0.0013
Ebitda-servizio debito/AN        0.0456   0.0144   3.1701 0.0015  0.0174  0.0738
Ln(AN)                          -1.0954   0.0913 -11.9941 0.0000 -1.2744 -0.9164
Ln(RIC)                          0.7856   0.0923   8.5089 0.0000  0.6046  0.9666
================================================================================
```

```
X= X.drop(['Patr netto/debiti totali'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
=================================================================================
Model:              Logit                          Pseudo R-squared: 0.162
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1717.8308
Date:               2020-11-16 12:42               BIC:              1843.3561
No. Observations:   31836                          Log-Likelihood:   -843.92
Df Model:           14                             LL-Null:          -1006.5
Df Residuals:       31821                          LLR p-value:      6.5056e-61
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
---------------------------------------------------------------------------------
                                 Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------------------
Utile corrente/ricavi           -0.0047   0.0035  -1.3270 0.1845 -0.0116  0.0022
ROE                             -0.0043   0.0016  -2.7134 0.0067 -0.0075 -0.0012
Val Agg Oper/Ricavi             -0.0142   0.0039  -3.6720 0.0002 -0.0217 -0.0066
servizi esterni/costi operativi -0.0211   0.0037  -5.7100 0.0000 -0.0283 -0.0138
Ricavi/AN                       -1.9752   0.2119  -9.3228 0.0000 -2.3904 -1.5599
gg magazz                       -0.0049   0.0013  -3.7417 0.0002 -0.0075 -0.0023
AC/PC                           -0.0011   0.0008  -1.3750 0.1691 -0.0027  0.0005
Riserve+utile/AN                -0.0267   0.0046  -5.8534 0.0000 -0.0356 -0.0177
Patr netto tang/AN               0.0020   0.0046   0.4395 0.6603 -0.0071  0.0111
Auto Lordo-comp straord/AN      -0.0276   0.0237  -1.1681 0.2428 -0.0740  0.0187
Deb finanziari (stimati)/VA      0.0008   0.0003   2.9521 0.0032  0.0003  0.0014
Debiti finanziari/EBITDA         0.0006   0.0003   2.0219 0.0432  0.0000  0.0013
Ebitda-servizio debito/AN        0.0443   0.0144   3.0836 0.0020  0.0162  0.0725
Ln(AN)                          -1.0946   0.0919 -11.9157 0.0000 -1.2746 -0.9145
Ln(RIC)                          0.7788   0.0928   8.3921 0.0000  0.5969  0.9606
=================================================================================
```

```
X= X.drop(['Patr netto tang/AN'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
=================================================================================
Model:              Logit                          Pseudo R-squared: 0.161
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1716.0229
Date:               2020-11-16 12:43               BIC:              1833.1799
No. Observations:   31836                          Log-Likelihood:   -844.01
Df Model:           13                             LL-Null:          -1006.5
Df Residuals:       31822                          LLR p-value:      1.3957e-61
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
---------------------------------------------------------------------------------
                                 Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------------------
Utile corrente/ricavi           -0.0046   0.0035  -1.3162 0.1881 -0.0115  0.0023
ROE                             -0.0040   0.0014  -2.8504 0.0044 -0.0067 -0.0012
Val Agg Oper/Ricavi             -0.0141   0.0039  -3.6644 0.0002 -0.0217 -0.0066
servizi esterni/costi operativi -0.0211   0.0037  -5.6989 0.0000 -0.0283 -0.0138
Ricavi/AN                       -1.9750   0.2118  -9.3242 0.0000 -2.3902 -1.5599
gg magazz                       -0.0049   0.0013  -3.7777 0.0002 -0.0075 -0.0024
AC/PC                           -0.0009   0.0007  -1.3560 0.1751 -0.0023  0.0004
Riserve+utile/AN                -0.0260   0.0043  -6.0408 0.0000 -0.0344 -0.0175
Auto Lordo-comp straord/AN      -0.0282   0.0237  -1.1876 0.2350 -0.0747  0.0183
Deb finanziari (stimati)/VA      0.0008   0.0003   2.9741 0.0029  0.0003  0.0014
Debiti finanziari/EBITDA         0.0007   0.0003   2.1077 0.0351  0.0000  0.0013
Ebitda-servizio debito/AN        0.0445   0.0144   3.0955 0.0020  0.0163  0.0727
Ln(AN)                          -1.0907   0.0916 -11.9126 0.0000 -1.2702 -0.9113
Ln(RIC)                          0.7768   0.0928   8.3711 0.0000  0.5949  0.9587
=================================================================================
```

```
X= X.drop(['Ebitda-servizio debito/AN'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
=================================================================================
Model:              Logit                          Pseudo R-squared: 0.156
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1724.1183
Date:               2020-11-16 12:44               BIC:              1832.9069
No. Observations:   31836                          Log-Likelihood:   -849.06
Df Model:           12                             LL-Null:          -1006.5
Df Residuals:       31823                          LLR p-value:      3.4845e-60
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
---------------------------------------------------------------------------------
                                 Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------------------
Utile corrente/ricavi           -0.0027   0.0035  -0.7938 0.4273 -0.0095  0.0040
ROE                             -0.0035   0.0014  -2.4615 0.0138 -0.0063 -0.0007
Val Agg Oper/Ricavi             -0.0141   0.0038  -3.6830 0.0002 -0.0216 -0.0066
servizi esterni/costi operativi -0.0215   0.0037  -5.8073 0.0000 -0.0287 -0.0142
Ricavi/AN                       -2.0377   0.2121  -9.6057 0.0000 -2.4534 -1.6219
gg magazz                       -0.0050   0.0013  -3.7831 0.0002 -0.0075 -0.0024
AC/PC                           -0.0009   0.0007  -1.3298 0.1836 -0.0023  0.0004
Riserve+utile/AN                -0.0234   0.0043  -5.5046 0.0000 -0.0318 -0.0151
Auto Lordo-comp straord/AN       0.0111   0.0176   0.6327 0.5269 -0.0233  0.0456
Deb finanziari (stimati)/VA      0.0005   0.0003   1.7729 0.0762 -0.0000  0.0010
Debiti finanziari/EBITDA         0.0006   0.0003   1.8647 0.0622 -0.0000  0.0012
Ln(AN)                          -1.0733   0.0915 -11.7296 0.0000 -1.2526 -0.8939
Ln(RIC)                          0.7574   0.0926   8.1820 0.0000  0.5760  0.9389
=================================================================================
```

```
X= X.drop(['Auto Lordo-comp straord/AN'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
=================================================================================
Model:              Logit                          Pseudo R-squared: 0.156
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1722.5079
Date:               2020-11-16 12:45               BIC:              1822.9281
No. Observations:   31836                          Log-Likelihood:   -849.25
Df Model:           11                             LL-Null:          -1006.5
Df Residuals:       31824                          LLR p-value:      7.6680e-61
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
---------------------------------------------------------------------------------
                                 Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------------------
Utile corrente/ricavi           -0.0024   0.0034  -0.6944 0.4875 -0.0091  0.0043
ROE                             -0.0034   0.0014  -2.4199 0.0155 -0.0062 -0.0007
Val Agg Oper/Ricavi             -0.0137   0.0038  -3.6294 0.0003 -0.0211 -0.0063
servizi esterni/costi operativi -0.0212   0.0037  -5.7594 0.0000 -0.0284 -0.0140
Ricavi/AN                       -2.0199   0.2104  -9.5994 0.0000 -2.4324 -1.6075
gg magazz                       -0.0050   0.0013  -3.7989 0.0001 -0.0076 -0.0024
AC/PC                           -0.0009   0.0007  -1.3077 0.1910 -0.0022  0.0004
Riserve+utile/AN                -0.0234   0.0043  -5.4986 0.0000 -0.0318 -0.0151
Deb finanziari (stimati)/VA      0.0004   0.0003   1.6836 0.0923 -0.0001  0.0009
Debiti finanziari/EBITDA         0.0005   0.0003   1.8034 0.0713 -0.0000  0.0010
Ln(AN)                          -1.0583   0.0887 -11.9345 0.0000 -1.2321 -0.8845
Ln(RIC)                          0.7476   0.0916   8.1643 0.0000  0.5682  0.9271
=================================================================================
```

```
X= X.drop(['servizi esterni/costi operativi'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
==========================================================================
Model:              Logit                          Pseudo R-squared: 0.139
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1754.3256
Date:               2020-11-16 12:46               BIC:              1846.3774
No. Observations:   31836                          Log-Likelihood:   -866.16
Df Model:           10                             LL-Null:          -1006.5
Df Residuals:       31825                          LLR p-value:      1.8654e-54
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
--------------------------------------------------------------------------
                             Coef.   Std.Err.     z      P>|z|    [0.025   0.975]
--------------------------------------------------------------------------
Utile corrente/ricavi       -0.0037    0.0036   -1.0295  0.3032  -0.0106   0.0033
ROE                         -0.0026    0.0014   -1.8785  0.0603  -0.0053   0.0001
Val Agg Oper/Ricavi         -0.0098    0.0039   -2.5047  0.0123  -0.0175  -0.0021
Ricavi/AN                   -2.3417    0.2131  -10.9863  0.0000  -2.7595  -1.9239
gg magazz                   -0.0037    0.0013   -2.8910  0.0038  -0.0062  -0.0012
AC/PC                       -0.0016    0.0007   -2.3013  0.0214  -0.0030  -0.0002
Riserve+utile/AN            -0.0212    0.0041   -5.1808  0.0000  -0.0293  -0.0132
Deb finanziari (stimati)/VA  0.0004    0.0003    1.7093  0.0874  -0.0001   0.0009
Debiti finanziari/EBITDA     0.0004    0.0003    1.5696  0.1165  -0.0001   0.0009
Ln(AN)                      -1.3550    0.0749  -18.0935  0.0000  -1.5017  -1.2082
Ln(RIC)                      1.0065    0.0834   12.0721  0.0000   0.8431   1.1699
==========================================================================
```

```
X= X.drop(['Ln(RIC)'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
==========================================================================
Model:              Logit                          Pseudo R-squared: 0.055
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1921.5655
Date:               2020-11-16 12:47               BIC:              2005.2491
No. Observations:   31836                          Log-Likelihood:   -950.78
Df Model:           9                              LL-Null:          -1006.5
Df Residuals:       31826                          LLR p-value:      7.4620e-20
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
--------------------------------------------------------------------------
                             Coef.   Std.Err.     z      P>|z|    [0.025   0.975]
--------------------------------------------------------------------------
Utile corrente/ricavi        0.0052    0.0030    1.7095  0.0874  -0.0008   0.0111
ROE                         -0.0022    0.0013   -1.7568  0.0790  -0.0047   0.0003
Val Agg Oper/Ricavi         -0.0070    0.0038   -1.8465  0.0648  -0.0145   0.0004
Ricavi/AN                   -0.6915    0.1353   -5.1111  0.0000  -0.9566  -0.4263
gg magazz                   -0.0011    0.0011   -0.9781  0.3280  -0.0033   0.0011
AC/PC                       -0.0038    0.0008   -5.0655  0.0000  -0.0053  -0.0023
Riserve+utile/AN            -0.0108    0.0037   -2.9520  0.0032  -0.0180  -0.0036
Deb finanziari (stimati)/VA -0.0005    0.0002   -2.1844  0.0289  -0.0010  -0.0001
Debiti finanziari/EBITDA     0.0005    0.0003    1.7228  0.0849  -0.0001   0.0010
Ln(AN)                      -0.5761    0.0328  -17.5530  0.0000  -0.6404  -0.5118
==========================================================================
```

```
X= X.drop(['Utile corrente/ricavi'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
=================================================================================
Model:              Logit                          Pseudo R-squared: 0.054
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1922.4330
Date:               2020-11-16 12:47               BIC:              1997.7482
No. Observations:   31836                          Log-Likelihood:   -952.22
Df Model:           8                              LL-Null:          -1006.5
Df Residuals:       31827                          LLR p-value:      7.4576e-20
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
---------------------------------------------------------------------------------
                             Coef.   Std.Err.     z      P>|z|    [0.025   0.975]
---------------------------------------------------------------------------------
ROE                         -0.0021    0.0013   -1.6397  0.1011  -0.0045   0.0004
Val Agg Oper/Ricavi         -0.0034    0.0031   -1.0974  0.2725  -0.0094   0.0027
Ricavi/AN                   -0.6708    0.1350   -4.9680  0.0000  -0.9355  -0.4062
gg magazz                   -0.0011    0.0011   -0.9951  0.3197  -0.0033   0.0011
AC/PC                       -0.0039    0.0008   -5.1326  0.0000  -0.0054  -0.0024
Riserve+utile/AN            -0.0103    0.0036   -2.8157  0.0049  -0.0174  -0.0031
Deb finanziari (stimati)/VA -0.0006    0.0002   -2.2998  0.0215  -0.0011  -0.0001
Debiti finanziari/EBITDA     0.0003    0.0003    1.2195  0.2226  -0.0002   0.0008
Ln(AN)                      -0.5895    0.0322  -18.3302  0.0000  -0.6525  -0.5265
=================================================================================
```

```
X= X.drop(['Deb finanziari (stimati)/VA'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
=================================================================================
Model:              Logit                          Pseudo R-squared: 0.051
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1926.0893
Date:               2020-11-16 12:48               BIC:              1993.0361
No. Observations:   31836                          Log-Likelihood:   -955.04
Df Model:           7                              LL-Null:          -1006.5
Df Residuals:       31828                          LLR p-value:      2.6851e-19
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
---------------------------------------------------------------------------------
                             Coef.   Std.Err.     z      P>|z|    [0.025   0.975]
---------------------------------------------------------------------------------
ROE                         -0.0016    0.0012   -1.2954  0.1952  -0.0040   0.0008
Val Agg Oper/Ricavi         -0.0046    0.0031   -1.5163  0.1294  -0.0106   0.0014
Ricavi/AN                   -0.5906    0.1278   -4.6220  0.0000  -0.8410  -0.3401
gg magazz                   -0.0011    0.0011   -0.9540  0.3401  -0.0032   0.0011
AC/PC                       -0.0041    0.0008   -5.4874  0.0000  -0.0056  -0.0027
Riserve+utile/AN            -0.0082    0.0035   -2.3495  0.0188  -0.0150  -0.0014
Debiti finanziari/EBITDA     0.0004    0.0002    1.5587  0.1191  -0.0001   0.0009
Ln(AN)                      -0.6180    0.0301  -20.5083  0.0000  -0.6771  -0.5590
=================================================================================
```

Il modello ha tutte le variabili entranti con un coefficiente corretto; non resta che effettuare l'ultima correzione andando a rimuovere, anche in questo caso una alla volta, quelle variabili che presentano un p-value > 5% (ovvero che al 95% di confidenza non sono significative):

```
X= X.drop(['gg magazz'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
================================================================================
Model:              Logit                          Pseudo R-squared: 0.051
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1925.0358
Date:               2020-11-16 12:49               BIC:              1983.6142
No. Observations:   31836                          Log-Likelihood:   -955.52
Df Model:           6                              LL-Null:          -1006.5
Df Residuals:       31829                          LLR p-value:      9.7075e-20
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
--------------------------------------------------------------------------------
                             Coef.   Std.Err.     z      P>|z|    [0.025   0.975]
--------------------------------------------------------------------------------
ROE                         -0.0015    0.0012   -1.2459  0.2128  -0.0039   0.0009
Val Agg Oper/Ricavi         -0.0047    0.0031   -1.5554  0.1199  -0.0107   0.0012
Ricavi/AN                   -0.5785    0.1276   -4.5335  0.0000  -0.8286  -0.3284
AC/PC                       -0.0041    0.0008   -5.4656  0.0000  -0.0056  -0.0027
Riserve+utile/AN            -0.0086    0.0034   -2.5065  0.0122  -0.0154  -0.0019
Debiti finanziari/EBITDA     0.0004    0.0002    1.5105  0.1309  -0.0001   0.0009
Ln(AN)                      -0.6260    0.0291  -21.4996  0.0000  -0.6830  -0.5689
================================================================================
```

```
X= X.drop(['ROE'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
================================================================================
Model:              Logit                          Pseudo R-squared: 0.050
Dependent Variable: FLAG di sana in liquid + anomala AIC:              1924.5881
Date:               2020-11-16 12:50               BIC:              1974.7983
No. Observations:   31836                          Log-Likelihood:   -956.29
Df Model:           5                              LL-Null:          -1006.5
Df Residuals:       31830                          LLR p-value:      4.3028e-20
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
--------------------------------------------------------------------------------
                             Coef.   Std.Err.     z      P>|z|    [0.025   0.975]
--------------------------------------------------------------------------------
Val Agg Oper/Ricavi         -0.0043    0.0030   -1.4312  0.1524  -0.0103   0.0016
Ricavi/AN                   -0.5815    0.1279   -4.5470  0.0000  -0.8321  -0.3308
AC/PC                       -0.0043    0.0007   -5.7302  0.0000  -0.0058  -0.0028
Riserve+utile/AN            -0.0117    0.0024   -4.7999  0.0000  -0.0165  -0.0069
Debiti finanziari/EBITDA     0.0005    0.0002    2.0996  0.0358   0.0000   0.0009
Ln(AN)                      -0.6182    0.0283  -21.8252  0.0000  -0.6738  -0.5627
================================================================================
```

```
X= X.drop(['Val Agg Oper/Ricavi'], axis=1)

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
==========================================================================
Model:              Logit                           Pseudo R-squared: 0.049
Dependent Variable: FLAG di sana in liquid + anomala AIC:             1924.7032
Date:               2020-11-16 12:50                BIC:              1966.5450
No. Observations:   31836                           Log-Likelihood:   -957.35
Df Model:           4                               LL-Null:          -1006.5
Df Residuals:       31831                           LLR p-value:      2.2535e-20
Converged:          1.0000                          Scale:            1.0000
No. Iterations:     10.0000
--------------------------------------------------------------------------
                             Coef.   Std.Err.     z      P>|z|    [0.025   0.975]
--------------------------------------------------------------------------
Ricavi/AN                   -0.6287    0.1239   -5.0739  0.0000  -0.8715  -0.3858
AC/PC                       -0.0043    0.0008   -5.7683  0.0000  -0.0058  -0.0029
Riserve+utile/AN            -0.0119    0.0024   -4.9625  0.0000  -0.0166  -0.0072
Debiti finanziari/EBITDA     0.0007    0.0002    4.3035  0.0000   0.0004   0.0010
Ln(AN)                      -0.6341    0.0261  -24.2762  0.0000  -0.6853  -0.5829
==========================================================================
```

Queste sono le 5 variabili e la y oggetto del modello di regressione logistica:

```
X
```

	Ricavi/AN	AC/PC	Riserve+utile/AN	Debiti finanziari/EBITDA	Ln(AN)
0	0.52	105.06	24.22	1000.00	11.10
1	0.68	138.75	24.26	21.34	11.10
2	0.82	129.64	21.81	73.91	11.10
3	0.90	142.72	20.98	18.14	11.10
4	1.27	130.84	20.45	7.94	11.10
...	...	...	...	...	...
31831	0.10	35.23	-58.12	46.08	4.39
31832	0.38	60.00	-58.12	46.08	4.39
31833	0.33	600.00	-22.22	46.08	4.39
31834	0.33	600.00	-22.22	35.84	4.39
31835	0.03	600.00	0.00	46.08	4.77

31836 rows × 5 columns

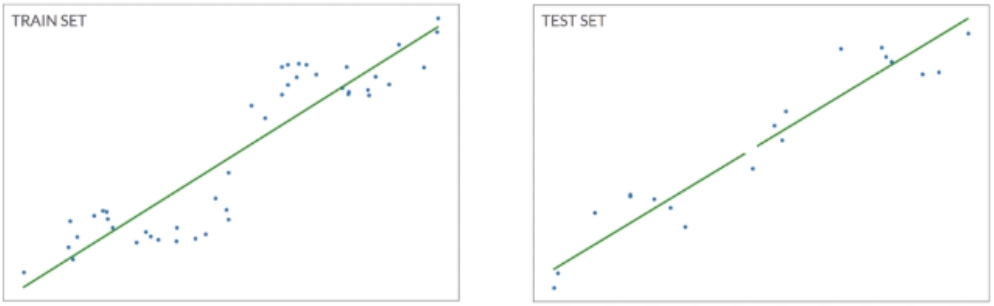
```
y #y_anno
```

	FLAG di sana in liquid + anomala
0	0
1	0
2	0
3	0
4	0
...	...
31831	0
31832	0
31833	0
31834	0
31835	0

31836 rows × 1 columns

Successivamente avviene la fase di *Data Preprocessing*, la quale comprende quattro particolari tecniche: *Split*, *Scaler*, *Hyperparameter optimization* e *Validation* del dataset (le ultime due sono state trattate nel capitolo precedente).

1. La prima consiste nel suddividere il dataset in due set differenti, *training* e *test* set:



La suddivisione del dataset in un set di addestramento e uno di test permette di ridurre l'overfitting e, di conseguenza, ottenere un modello in grado di generalizzare su nuovi dati i pattern individuati durante l'addestramento; infatti, al fine di ottenere un modello predittivo stabile è fondamentale eseguire il test su dati totalmente sconosciuti al modello stesso (dati non presenti nel set di training). Sicuramente il set di addestramento deve contenere un numero di osservazioni di gran lunga superiore a quello di test e, in linea generale, per dataset di dimensioni ristrette (tra i 1000-10000 esempi totali) è consigliato uno split 70-30 oppure 80-20. Al contrario, disponendo di un dataset composto da milioni di esempi, il solo 2% allocato al test set può essere sufficiente per testare il modello.

Nel caso in esame, avendo a disposizione circa 32000 osservazioni, lo split è stato effettuato con un rapporto 80-20 (*test_size=0.2*) attraverso la funzione *train_test_split*:

```
#split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_state=123456789)
```

dove:

- (*X_train*, *y_train*) è il set di addestramento;
- (*X_test*, *y_test*) è il set di test;
- *random_state* è un parametro che espone una serie di metodi per la generazione di numeri casuali tratti da una varietà di distribuzioni di probabilità; oltre agli argomenti specifici delle varie distribuzioni, ogni metodo accetta una dimensione dell'argomento che viene impostata attraverso il random_state; utilizzando per ogni modello sempre lo stesso valore del random_state (numero o serie di numeri), si ha la garanzia che venga utilizzato, per ogni modello, lo stesso metodo per splittare il dataset; dunque, per dataset aventi lo stesso numero di osservazioni vi è la certezza che lo split avvenga sempre nello stesso punto, qualunque modello si utilizzi. Questo parametro è molto importante per andare a confrontare le performance tra i modelli in quanto i vari X_train, X_test, y_train e y_test comprenderanno sempre non solo lo stesso numero totale di osservazioni ma anche le stesse singole osservazioni.

```
X_train.shape
```

```
(25468, 5)
```

```
y_train.shape
```

```
(25468, 1)
```

```
X_test.shape
```

```
(6368, 5)
```

```
y_test.shape
```

```
(6368, 1)
```

Dunque, per ogni modello sviluppato si avrà uno split del set di addestramento e di test rispettivamente di 25468 e 6368

osservazioni totali, conservando inoltre, da un modello all'altro, esattamente le stesse osservazioni che li compongono.

2. Per quanto concerne lo Scaler, nella maggior parte dei modelli di Machine e Deep Learning risulta fondamentale avere dei dati appartenenti alla stessa scala di valori: dato che alcune colonne (features) possono avere valori molto più alti o molto più bassi di altre, c'è il forte rischio che l'algoritmo di apprendimento dia più importanza alle variabili con un range di valori più ampio. Esistono due metodi per portare i dati a una stessa scala:
   - *Normalizzazione*: permette la conversione di tutti i numeri in un range tra 0 e 1. Dato un vettore qualsiasi contenente n elementi:
     $$x = [x^{(1)}, \quad x^{(2)}, \quad x^{(3)}, \dots, \quad x^{(n)}]$$
     la normalizzazione avviene applicando la seguente formula:
     $$\boldsymbol{x_{norm}^{(i)}} = \frac{\boldsymbol{x^{(i)} - x_{min}}}{\boldsymbol{x_{max} - x_{min}}}$$
     applicata ad ogni valore del vettore.

   - *Standardizzazione*: trasforma il dataset in una distribuzione normale, ovvero a media nulla e deviazione standard pari a 1; questo significa che i valori avranno un range tra -1 e 1. Dato un vettore qualsiasi contenente n elementi:
     $$x = [x^{(1)}, \quad x^{(2)}, \quad x^{(3)}, \dots, \quad x^{(n)}]$$
     la standardizzazione si esegue mediante la formula per ogni vettore colonna del dataset:
     $$\boldsymbol{x_{std}^{(i)}} = \frac{\boldsymbol{x^{(i)} - x_{mean}}}{\boldsymbol{x_{sd}}}$$

dove x_{mean} e x_{sd} sono rispettivamente il valore medio e deviazione standard del vettore.

La scelta di una o dell'altra dipende dal caso pratico in esame; ci sono modelli lineari per i quali avere i dati distribuiti in forma normale può agevolare l'apprendimento e, inoltre, l'utilizzo della standardizzazione consente di mantenere le informazioni sugli outliers. Dato che uno dei modelli successivi sarà una rete neurale e che questa necessita di dati in input in un range tra 0 e 1, si è preferito utilizzare la normalizzazione (classe MinMaxScaler) al fine di confrontare i modelli, al netto dei metodi. Si applica, dunque, lo Scaler sui nuovi set X_train X_test:

```
ss= MinMaxScaler()
X_train= ss.fit_transform(X_train)
X_test= ss.transform(X_test)
```

```
X_train
```

```
array([[0.49065421, 0.09821697, 0.49134181, 0.00866061, 0.67064083],
       [0.31775701, 0.18511961, 0.61288469, 0.01973138, 1.        ],
       [0.73831776, 0.09566726, 0.48042675, 0.01343094, 0.53651267],
       ...,
       [0.6635514 , 0.1699807 , 0.72137874, 0.00409029, 0.50670641],
       [0.28504673, 0.38681233, 0.71456709, 0.        , 0.63934426],
       [0.57476636, 0.43886892, 0.78588428, 0.        , 0.46497765]])
```

```
X_test
```

```
array([[1.72897196e-01, 2.11023957e-01, 4.29298318e-01, 7.00049003e-05,
        0.00000000e+00],
       [6.07476636e-02, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00],
       [2.75700935e-01, 2.16937869e-01, 5.14895363e-01, 1.38009661e-02,
        4.36661699e-01],
       ...,
       [1.40186916e-02, 0.00000000e+00, 0.00000000e+00, 4.88334183e-02,
        0.00000000e+00],
       [6.68224299e-01, 3.10745967e-02, 4.88469430e-01, 1.03007211e-02,
        1.89269747e-01],
       [1.21495327e-01, 1.00000000e+00, 9.35330324e-01, 7.24050684e-03,
        4.67958271e-01]])
```

Successivamente si procede con la fase di tuning dei parametri mediante la tecnica *Random Search*:

```
from sklearn.model_selection import RandomizedSearchCV

logisticRegr = LogisticRegression(class_weight='balanced', fit_intercept=True,intercept_scaling=1,
                                  max_iter=100, multi_class='ovr', random_state=123456789,
                                  tol=0.0001, verbose=0)

space = dict()
space['solver'] = ['lbfgs', 'liblinear']
space['penalty'] = ['none', 'l1', 'l2']

grid = RandomizedSearchCV(logisticRegr, space, scoring='recall', n_jobs=-1, cv=3)

random_result = grid.fit(X_train, y_train)
```

Dunque, si imposta la funzione della regressione logistica (*LogisticRegression*), importata dalla libreria scikit-learn, con tutti i parametri che si decide a priori di mantenere fissi, ovvero quelli che non saranno soggetti al processo di ottimizzazione:

- *class_weight* (default=None): si associa un peso ad ogni classe nella forma {class_label: weight}; se non dato, tutte le classi avranno peso uguale a 1;
- *fit_intercept* (bool, default=True): se si specifica una costante, ovvero il bias o intercetta, deve essere aggiunta alla funzione decisionale;
- *intercept_scaling* (float, default=1): utile solamente nel caso in cui il solutore sia il "liblinear" e il self.fit_intercept è impostato a True;
- *max_iter* (int, default=100): massimo numero di iterazioni necessarie per la convergenza dei solutori;
- *multi_class* ('auto', 'ovr', 'multinomial', default='auto'): "auto" seleziona "ovr" se i dati sono binari oppure se il solver è il "liblinear", altrimenti seleziona "multinomial". Quindi, se il problema è di classificazione binaria il parametro sarà "ovr";
- *random_state*: settato sempre alla stessa sequenza di numeri;
- *tol* (float, default=0.0001): da utilizzare quando si vuole fermare in anticipo l'algoritmo nel caso in cui ad ogni iterazione la funzione di costo non diminuisce di un piccolo valore prefissato; dunque, tale valore è la tolleranza per i criteri di arresto;

- *verbose* (int, default=0): per i solutori "liblinear" e "lbfgs" impostarlo ad un valore positivo o lasciarlo al suo valore di default.

A questo punto viene definito uno spazio di ricerca (*space*); questo può essere inteso geometricamente come un volume n-dimensionale in cui ogni iperparametro è una dimensione diversa e la scala della dimensione è rappresentata dai valori che l'iperparametro può assumere (valore reale, intero o categoriale). Si è deciso di ottimizzare 2 iperparametri della regressione logistica:

- *solver* ('lbfgs', 'liblinear', 'sag', 'saga', default='lbfgs'): rappresenta l'algoritmo utilizzato per ottimizzare il problema. Per dataset ristretti la "liblinear" è la scelta migliore, mentre "sag" e "saga" sono quelli più veloci per quelli di grandi dimensioni. Per i problemi multiclasse solo il "newton-cg", "sag", "saga" e "lbfgs" riescono a gestire la perdita multinomiale, la penalità L2 o nessuna penalità. Al contrario, la "liblinear" e "saga" riescono a gestire la penalità L1;
- *penalty* ('none', 'l1', 'l2', 'elasticnet', default='none'): è il parametro che introduce le regolarizzazioni.

Lo spazio di ricerca della combinazione ottima è stato ridotto a ['lbfgs', 'liblinear'] per il solver e ['none', 'l1', 'l2'] per la penalty. Oltre il tuning degli iperparametri, la classe RandomizedSearchCV esegue una prima cross validation con un numero di folds pari a 3 (valore di default). I risultati ottenuti al termine dell'addestramento (*fit*) della Random Search sono:

```
# report the best configuration
print("Best: %f using %s" % (random_result.best_score_, random_result.best_params_))
```

```
Best: 0.787879 using {'solver': 'lbfgs', 'penalty': 'none'}
```

La miglior combinazione[21], dunque, risulta 'lbfgs' come solver e nessuna regolarizzazione.

Quindi, viene eseguita separatamente la *Validation* sul seguente modello per testarne la robustezza:

```
from sklearn.model_selection import cross_val_score

logisticRegr = LogisticRegression(class_weight='balanced', fit_intercept=True,intercept_scaling=1,
                                  max_iter=100, multi_class='ovr', random_state=123456789,
                                  solver='lbfgs', tol=0.0001, verbose=0)

scores = cross_val_score(logisticRegr, X_train, y_train, cv=10)

for fold,score in enumerate(scores):
    print("Fold %d score=%.4f" % (fold+1,score))

print("\nValidation Accuracy = %.2f" % scores.mean())
```

```
Fold 1 score=0.8327
Fold 2 score=0.8449
Fold 3 score=0.8418
Fold 4 score=0.8339
Fold 5 score=0.8320
Fold 6 score=0.8445
Fold 7 score=0.8418
Fold 8 score=0.8418
Fold 9 score=0.8303
Fold 10 score=0.8256

Validation Accuracy = 0.84
```

Si imposta nuovamente il modello di regressione logistica con i vari iperparametri ottimizzati (ma da convalidare) e, attraverso la funzione *cross_val_score* importata da scikit-learn, si creano i 10 folds (cv) per effettuare l'addestramento su ognuno di essi. Con tale funzione, la *10-fold cross validation* viene utilizzata come fosse una funzione di scoring, con la variabile *scores* che conterrà le 10 accuracy dei 10 modelli costruiti tramite cross validation; l'accuracy complessiva del modello è la media delle singole accuracy. Come è facile notare, il modello risulta convalidato.

In questo primo modello la fase di ottimizzazione non è stata molto incidente in quanto il parametro che ha determinato maggiormente i risultati ottenuti è

[21] La fase di tuning è stata eseguita sulla base della 'recall' (ovvero in base al parametro *scoring*), dunque la classe Random Search ha selezionato la combinazione di parametri che massimizza tale indicatore, ma l'attributo *.best_score_* ritorna la recall media ottenuta dalla migliore combinazione nei 3 folds.

stato il *class_weight*. In particolare, il dataset utilizzato risulta essere molto "sbilanciato" in quanto la target *anno* contiene un numero di 1 (160) decisamente inferiore al numero di 0 (31676); questo significa che nel momento in cui il modello osserva un *1*, essendo questo un evento raro, deve avere una maggior rilevanza rispetto al caso comune in cui osserva uno *0*; dunque, è chiaro che le classi *0/1* non possono avere lo stesso peso. Impostando il parametro *class_weight* a *"balanced"*, il modello utilizza i valori di y (target) per regolare automaticamente i pesi da assegnare ad ogni classe mediante l'applicazione della seguente formula:

$$\frac{\boldsymbol{n_{samples}}}{\boldsymbol{n_{classes} \cdot np.bincount(y)}}$$

In un problema di classificazione binaria la formula produrrà un vettore di due componenti nel quale il peso di una classe sarà tanto maggiore quanto minore è la frequenza del valore in questione all'interno della target y; dato che l'addestramento avviene solo nel training set, nel caso in esame ($n_{samples\_train}$ = 25468, $n_{classes}$ = 2, np.count(y_train) = [25336 ; 132]) si ottiene:

{peso_classe_0 ; peso_classe_1} = {0.5026 ; 96.47}

come da aspettarsi, il peso della classe delle anomale è enormemente superiore a quello della classe delle sane.

Adesso che il modello risulta ottimizzato e convalidato è necessario rieseguire sempre l'addestramento sull'intero training set, prima di testarlo sul test set:

```
logisticRegr.fit(X_train, y_train)
```

```
LogisticRegression(class_weight='balanced', multi_class='ovr',
                   random_state=123456789)
```

Si effettuano le predizioni su tale *set di addestramento*, in termini sia di *valore binario* sia in *probabilità* [prob 0, prob 1]:

```
# Perform prediction using the train dataset
y_pred = logisticRegr.predict(X_train)
y_pred_proba = logisticRegr.predict_proba(X_train)

print(y_pred)
print('\n')
print(y_pred_proba)
```

```
[0 0 0 ... 0 0 0]


[[0.76920832 0.23079168]
 [0.85128337 0.14871663]
 [0.75105781 0.24894219]
 ...
 [0.86959033 0.13040967]
 [0.85423774 0.14576226]
 [0.87123497 0.12876503]]
```

I risultati mostrano le prime e le ultime 3 osservazioni del set di addestramento le quali risultano tutte sane (probabilità *0* maggiore di quella *1*).

A questo punto si procede con il disegnare la Matrice di Confusione *(Confusion Matrix)*, ovvero lo strumento che permette di misurare l'efficacia di un modello di Machine Learning, mostrandone la quantità e tipologia di errori di predizione da esso commessi.

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

In altre parole, essa è utilizzata per valutare le prestazioni di un modello che tratta specificatamente problemi di classificazione; disponendo di 2 classi, la matrice di confusione si presenta con 4 diverse combinazioni di valori effettivi e previsti:

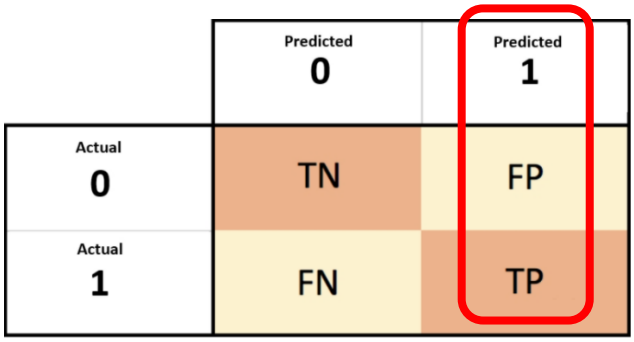
- *True Positive* (TP): l'osservazione appartiene alla classe delle anomale (1) e il modello la classifica correttamente.
- *True Negative* (TN): l'osservazione appartiene alla classe delle sane (0) e il modello la classifica correttamente.
- *False Positive* (FP): il modello predice positivo un esempio che in realtà è negativo (detto *errore di 1° specie*).

- *False Negative* (FN): il modello predice negativo un esempio che in realtà è positivo (detto *errore di 2° specie*).

È evidente che quest'ultimi due errori non sempre hanno lo stesso peso dato che, per determinati tipi di analisi sbagliare a classificare come negativa un'osservazione positiva risulta molto più grave rispetto al caso contrario; basti pensare al caso di un'analisi medica in cui definire un paziente negativo ad una certa malattia quando in realtà è positivo rischia di compromettere gravemente la salute futura di tale paziente, mentre l'errore contrario può essere riscontrato e corretto nelle analisi successive effettuate dal paziente "falso positivo".

La matrice di confusione è necessaria per andare a calcolare alcune tra le principali metriche di valutazione:

- *Precision*:

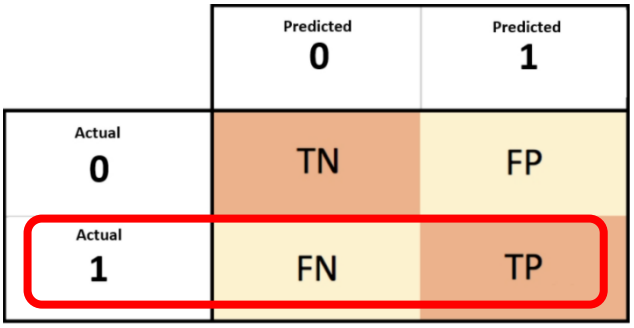


$$\boldsymbol{Precision = \frac{TP}{TP + FP}}$$

La precisione risponde alla domanda: "Di tutte le osservazioni positive che il modello ha previsto correttamente, quante sono effettivamente positive?"; infatti, il denominatore rappresenta il totale degli esempi positivi predetti. Quando i costi relativi al falso positivo sono alti viene utilizzata questa metrica per stabilire il modello definitivo. Può essere il caso, ad esempio, del rilevamento spam nella e-mail; avere un falso positivo significa che una mail

"non spam" è stata classificata come spam, quindi il costo che ne deriva è decisamente alto in quanto l'utente rischia di perdere e-mail molto importanti.

- *Recall*:



$$\boldsymbol{Recall = \frac{TP}{TP + FN}}$$

La recall risponde alla domanda: "Di tutte le osservazioni realmente positive, il modello quante ne ha classificate correttamente?"; si utilizza questa metrica per individuare il modello migliore quando il problema ha un elevato costo associato al caso falso negativo (come nell'esempio del paziente descritto in precedenza).

- *f1 – score*:

$$\boldsymbol{f1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}}$$

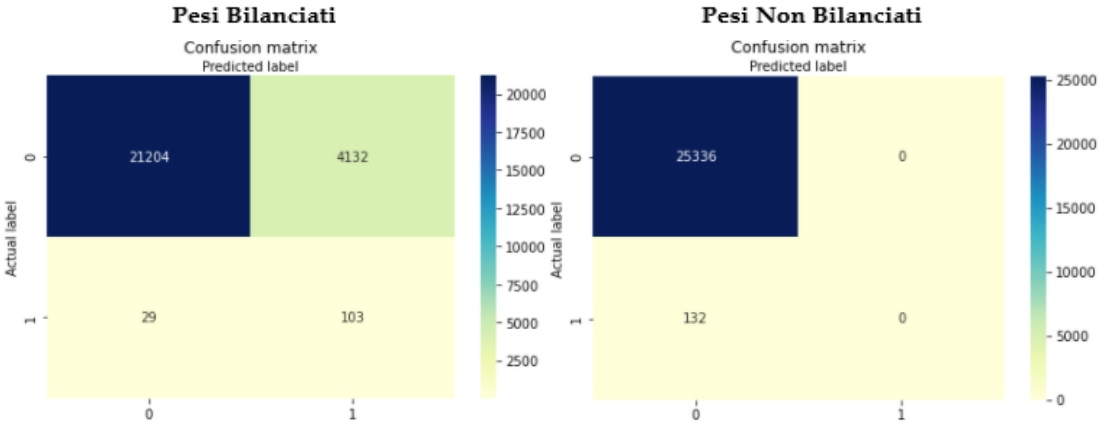
La metrica f1 viene utilizzata nel momento in cui si riscontrano forti difficoltà a confrontare due modelli con precisione e recall troppo discordanti (bassa precisione/alta recall o viceversa); essa con una media armonica tiene conto di entrambe penalizzando maggiormente i valori estremi.

- *Accuracy*:

L'accuratezza risponde alla domanda: "Di tutte le osservazioni, quante il modello ne ha classificate correttamente?"; indica, quindi, in quale percentuale le predizioni effettuate sono corrette.

In questo progetto di tesi riguardante un'analisi di credit scoring, l'errore più grave è sicuramente quello di classificare come sana (classe 0) un'osservazione che in realtà risulta anomala (classe 1), ovvero l'errore di 2° specie; avendo, dunque, come principale problema quello dei "falsi negativi", la metrica che deve essere presa maggiormente in considerazione è la *recall*. Tale considerazione vale ancora di più in questo primo modello in cui la target presenta solo lo 0.5% delle osservazioni anomale (160 su 31836). È proprio per questo motivo che è stato così determinante l'aver settato il parametro *class_weight = 'balanced'* in quanto ha permesso al modello di individuare il più alto numero possibile di eventi anomali, nonostante la loro rarità all'interno della target, anche a scapito di registrare un aumento sull'errore di 1° specie ("falsi positivi"). Di seguito sono riportati i risultati ottenuti sul training set, *con* e *senza i pesi bilanciati*:

```
# Show the Confusion Matrix

confusion_matrix_array= confusion_matrix(y_train, y_pred)
class_names=[0,1]
fig, ax= plt.subplots()
tick_marks= np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(confusion_matrix_array), annot=True, cmap='YlGnBu', fmt='g')
ax.xaxis.set_label_position('top')

plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```



```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.84	0.91	25336
1	0.02	0.78	0.05	132
accuracy			0.84	25468
macro avg	0.51	0.81	0.48	25468
weighted avg	0.99	0.84	0.91	25468

```
print(classification_report(y_train, y_pred))
```

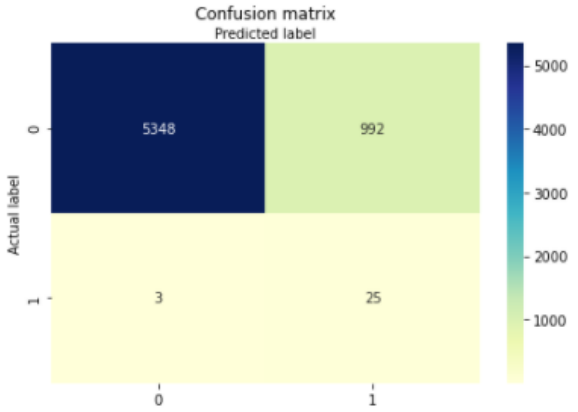
	precision	recall	f1-score	support
0	0.99	1.00	1.00	25336
1	0.00	0.00	0.00	132
accuracy			0.99	25468
macro avg	0.50	0.50	0.50	25468
weighted avg	0.99	0.99	0.99	25468

Dal risultato in figura è facile comprendere l'importanza di avere i pesi delle due classi bilanciati: nonostante si ottenga una precisione e accuracy del 99%, il caso *non bilanciato* non è minimamente accettabile in quanto il modello non riesce a individuare neanche una delle 132 osservazioni anomale presenti nel training set; al contrario, quando la risposta reale (actual label) è 0 (sana), il modello *bilanciato* predice correttamente 21204 osservazioni, sbagliandone 4132 (errore di 1° specie), mentre quando è 1 (anomala) si ha 103 esempi classificati correttamente e 29 falsi negativi (errore di 2° specie). Quindi, i valori della recall sono $\left[{}^{21204}/_{(21204+4132)} \cong 0.84\right]$ per la classe 0 e $\left[{}^{103}/_{(103+29)} \cong 0.78\right]$ per la classe 1. In definitiva, con un set di addestramento composto da 25468 osservazioni, di cui 25336 appartenenti alla classe delle sane e 132 alle anomale, questo modello classifica un'osservazione correttamente l'84% delle volte (accuracy).

Dopo aver calcolato le performance sul training set, vengono ripetute le stesse operazioni verificando il modello (scaturito dal training) sul *test set*; si effettuano le predizioni:

```
# Perform prediction using the test dataset
y_pred = logisticRegr.predict(X_test)
y_pred_proba = logisticRegr.predict_proba(X_test)

print(y_pred)
print('\n')
print(y_pred_proba)
```

```
[0 1 0 ... 1 0 0]


[[0.61017796 0.38982204]
 [0.27953591 0.72046409]
 [0.73446061 0.26553939]
 ...
 [0.26023817 0.73976183]
 [0.72101108 0.27898892]
 [0.87951572 0.12048428]]
```




```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.84	0.91	6340
1	0.02	0.89	0.05	28
accuracy			0.84	6368
macro avg	0.51	0.87	0.48	6368
weighted avg	1.00	0.84	0.91	6368

Il test set, composto da 6368 esempi, ha un numero di casi sani e anomali rispettivamente di 6340 e 28. Di quest'ultimi, il modello è in grado di individuarne 25, registrando, quindi, una performance addirittura superiore a quella del training set (89% di recall sulla classe 1); la situazione rimane inalterata, invece, per quanto concerne la classe 0.

In definitiva, si può affermare che questo modello (con target *anno*) è accettabile nel computo di un'analisi di credit scoring in quanto abbina buoni risultati a una discreta capacità di generalizzazione (non soffre di overfitting)

sui nuovi dati del test set, producendo un'accuratezza invariata rispetto a quella calcolata nel set di addestramento.

(target *società*)

```
import pandas as pd
import numpy as np

pd.set_option('display.max_columns',None)

credit= pd.read_csv(r'C:\Users\utente\Desktop\credit_completo_società.csv', encoding='latin1', sep=';')
credit.head()
```

	n.osservazioni (sequenza dopo ordinamento date dei bilanci)	n.società	RAGIONE SOCIALE	anno di riferimento del bilancio	FLAG di sana in liquid + anomala (società)	EBITDA/Ricavi	EBIT/ricavi	Utile corrente/ricavi	Risultato netto rettif/ricavi	Risult ne rettif/
0	1	1	ILVA S.P.A.	2009	1	-3.44	-13.47	-8.65	-13.16	-6
1	2	1	ILVA S.P.A.	2010	1	2.72	-5.56	1.43	0.93	0
2	3	1	ILVA S.P.A.	2011	1	0.64	-5.86	-6.01	-0.58	-0
3	4	2	MARCEGAGLIA S.R.L.	2009	0	6.16	3.15	1.54	-0.33	-0
4	5	2	MARCEGAGLIA S.R.L.	2010	0	5.88	3.03	2.48	0.78	0

```
credit['FLAG di sana in liquid + anomala (società)'].value_counts()
#il label 1 rappresenta il 15%, mentre con la y_anno era solo lo 0.5%
```

```
0    26979
1     4857
Name: FLAG di sana in liquid + anomala (società), dtype: int64
```

Il nuovo modello utilizza come risposta il flag delle *società* [*FLAG di sana in liquid + anomala (società)*]; di conseguenza si avrà a disposizione un numero di osservazioni positive pari a 4857. Il cambio della target è la sola differenza rispetto al modello precedente, per il resto si segue esattamente la stessa procedura (compresa la scelta di avere i pesi delle classi bilanciati). Il modello risultante è:

```
==========================================================================================
Model:                  Logit                                Pseudo R-squared: 0.291
Dependent Variable:     FLAG di sana in liquid + anomala (società) AIC:              19296.2274
Date:                   2020-11-16 14:45                     BIC:                19371.5426
No. Observations:       31836                                Log-Likelihood:     -9639.1
Df Model:               8                                    LL-Null:            -13598.
Df Residuals:           31827                                LLR p-value:        0.0000
Converged:              1.0000                               Scale:              1.0000
No. Iterations:         7.0000
------------------------------------------------------------------------------------------
                              Coef.     Std.Err.       z        P>|z|      [0.025     0.975]
------------------------------------------------------------------------------------------
ROE                         -0.0045       0.0003   -13.7966    0.0000    -0.0052    -0.0039
Val Agg Oper/Ricavi         -0.0081       0.0008    -9.8232    0.0000    -0.0097    -0.0064
gg magazz                   -0.0008       0.0003    -2.6805    0.0074    -0.0013    -0.0002
Riserve+utile/AN            -0.0175       0.0009   -19.9005    0.0000    -0.0192    -0.0158
OFN/EBIT                     0.0006       0.0002     3.3789    0.0007     0.0003     0.0010
Deb totali/VA                0.0004       0.0000     9.8958    0.0000     0.0003     0.0005
Debiti finanziari/EBITDA     0.0004       0.0001     4.8484    0.0000     0.0002     0.0005
Ln(AN)                      -0.0747       0.0125    -5.9490    0.0000    -0.0992    -0.0501
Ln(RIC)                     -0.2036       0.0118   -17.2441    0.0000    -0.2268    -0.1805
==========================================================================================
```

```
X
```

	ROE	Val Agg Oper/Ricavi	gg magazz	Riserve+utile/AN	OFN/EBIT	Deb totali/VA	Debiti finanziari/EBITDA	Ln(AN)	Ln(RIC)
0	-19.40	11.48	161.21	24.22	400.00	879.62	1000.00	11.10	11.11
1	1.82	15.32	140.68	24.26	400.00	506.47	21.34	11.10	11.11
2	-1.50	10.96	116.49	21.81	400.00	637.76	73.91	11.10	11.11
3	-1.00	13.91	154.54	20.98	51.29	531.52	18.14	11.10	11.11
4	3.47	11.51	125.27	20.45	17.98	463.24	7.94	11.10	11.11
...	...	...	...	...	...	...	...	...	...
31831	-200.00	33.33	0.00	-58.12	100.00	1756.72	46.08	4.39	1.10
31832	-200.00	33.33	226.23	-58.12	100.00	1000.00	46.08	4.39	1.10
31833	0.00	33.33	0.00	-22.22	100.00	100.00	46.08	4.39	1.10
31834	-12.50	33.33	0.00	-22.22	400.00	100.00	35.84	4.39	1.10
31835	0.00	33.33	0.00	0.00	100.00	100.00	46.08	4.77	1.10

31836 rows × 9 columns

```
y #y_società
```

	FLAG di sana in liquid + anomala (società)
0	1
1	1
2	1
3	0
4	0
...	...
31831	0
31832	0
31833	0
31834	0
31835	1

31836 rows × 1 columns

Da notare è il fatto che, mentre nel modello precedente una delle features entranti era "*Ricavi/AN*", utilizzando la target *società* il modello preferisce mantenere entrambe le variabili di dimensione aziendale "*Ln(AN)*" e "*Ln(RIC)*", interpretandole come "proxy" del turnover, invece di inserire direttamente tale variabile.

Utilizzando gli stessi comandi (e modalità) visti in precedenza si effettua lo split, lo scaler e il tuning:

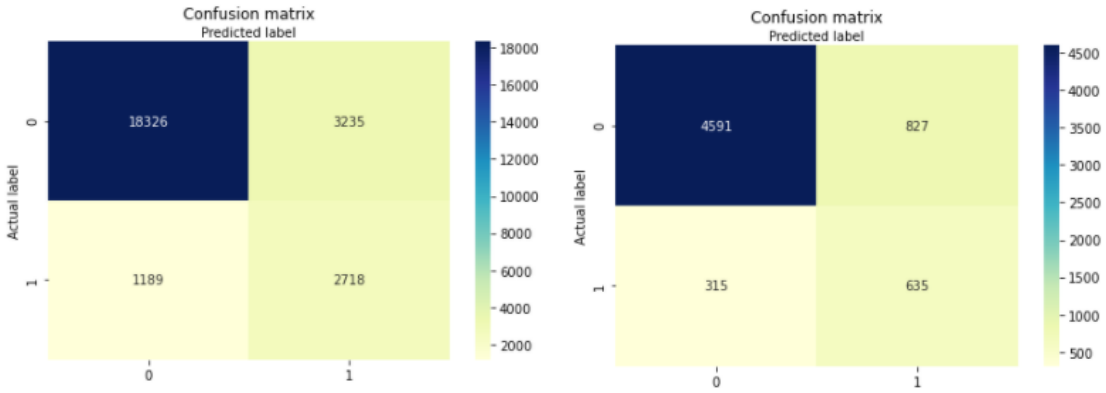
```
Best: 0.696695 using {'solver': 'liblinear', 'penalty': 'l1'}
```

Con la target *società* cambiano i due valori ottimizzati di solver e penalty.

```
from sklearn.model_selection import cross_val_score

logisticRegr = LogisticRegression(class_weight='balanced', fit_intercept=True,intercept_scaling=1,
                                  max_iter=100, multi_class='ovr', random_state=123456789,
                                  solver='liblinear', penalty='l1', tol=0.0001, verbose=0)

scores = cross_val_score(logisticRegr, X_train, y_train, cv=10)

for fold,score in enumerate(scores):
    print("Fold %d score=%.4f" % (fold+1,score))

print("\nValidation Accuracy = %.2f" % scores.mean())
```

```
Fold 1 score=0.8312
Fold 2 score=0.8178
Fold 3 score=0.8324
Fold 4 score=0.8292
Fold 5 score=0.8331
Fold 6 score=0.8261
Fold 7 score=0.8182
Fold 8 score=0.8214
Fold 9 score=0.8130
Fold 10 score=0.8390

Validation Accuracy = 0.83
```

Tale modello viene comunque convalidato. Di seguito i risultati:



```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	0.94	0.85	0.89	21561
1	0.46	0.70	0.55	3907
accuracy			0.83	25468
macro avg	0.70	0.77	0.72	25468
weighted avg	0.87	0.83	0.84	25468

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.94	0.85	0.89	5418
1	0.43	0.67	0.53	950
accuracy			0.82	6368
macro avg	0.69	0.76	0.71	6368
weighted avg	0.86	0.82	0.84	6368

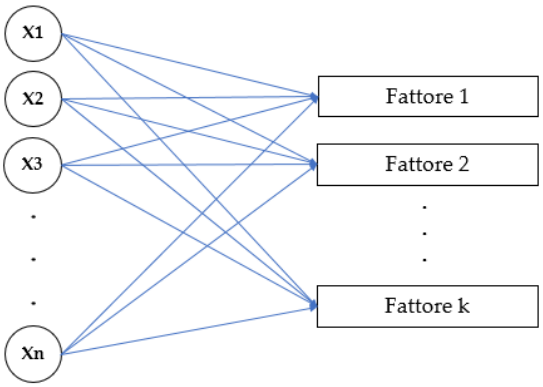
Lo split del dataset, grazie al *random_state*, è rimasto invariato a 25468 osservazioni nel train e 6368 nel test. I risultati mostrano una precisione sulla classe 1 decisamente aumentata rispetto al modello precedente e, in generale, uno score f1 molto più proporzionato; passando dal train al test set non ci sono particolari cambiamenti oltre a una perdita del 3% sulla recall della classe delle anomale. Dunque, questo modello, che registra un'accuracy dell'82%, risulta in confronto all'altro sicuramente più equilibrato ma con una performance decisamente più scarsa nella metrica principale per il credit scoring.

Regressione Logistica – Analisi Fattoriale

Data la numerosità delle features presenti all'interno del dataset iniziale, nei successivi modelli di regressione logistica si è utilizzata una particolare tecnica della data analysis chiamata *analisi fattoriale* (FA); di uso comune in molti ambiti come le ricerche di mercato (ad esempio, identificare la tipologia di clienti più sensibili al prezzo), finanza e pubblicità, questa cerca di agevolare l'interpretazione dei dati, riducendo un numero elevato di variabili in un numero inferiore di *fattori*. In particolare, l'analisi fattoriale permette di evidenziare l'esistenza di una struttura di fattori, non misurabili direttamente, all'interno di un insieme di variabili direttamente osservabili. Tali fattori, al loro interno, sono caratterizzati da una propria coerenza teorica, ovvero non si limitano a rappresentare l'esistenza di un'associazione statistica tra variabili

che li compongono ma devono anche essere interpretabili dal punto di vista scientifico-razionale[22].

Graficamente, partendo da un certo numero *n* di variabili [X_1, X_2, …, X_n], con l'analisi fattoriale si vanno a creare, tramite una loro combinazione lineare, un numero significativamente inferiore di *fattori* che dovranno spiegare una particolare quantità di varianza delle variabili stesse: (k < n)



$$\boldsymbol{y_i = \beta_{i0} + \beta_{i1} F_1 + \beta_{i2} F_2 + \cdots + \beta_{ik} F_k}$$

Le assunzioni per questo tipo di analisi sono:

- Non devono essere presenti outliers;
- La dimensione del campione deve essere maggiore di quella del fattore;
- Non ci dovrebbe essere *multicollinearità perfetta* tra le variabili; in un modello composto da X_n variabili, si ha multicollinearità perfetta quando due o più variabili sono tra loro perfettamente correlate, ovvero $X_i = \alpha + \beta X_j$. In questo caso la matrice $[X^T \cdot X]$ sarà singolare (determinante nullo) e, conseguentemente, lo stimatore $\hat{\beta}$ non offrirà stime attendibili.
- Non ci dovrebbe essere *omoschedasticità* tra le variabili, ovvero le features all'interno del dataset non devono avere tutte la stessa varianza.

[22] https://it.wikipedia.org/wiki/Analisi_fattoriale

[1° versione] (target *anno*)

Si comincia con lo stesso dataset base degli altri modelli, ovvero quello da 31836 osservazioni e 46 features + 1 target (*anno*). Vengono creati due array numpy al fine di dividere il dataset in X e y:

```
X= credit.drop('FLAG di sana in liquid + anomala', axis=1).values
y= credit['FLAG di sana in liquid + anomala'].values
```

```
cols_in=['EBITDA/Ricavi', 'EBIT/ricavi',
       'Utile corrente/ricavi', 'Risultato netto rettif/ricavi',
       'Risultato netto rettif/AN', 'EBITDA/AN', 'ROA', 'ROE',
       'ROE ante imposte', 'Val Agg Oper/Ricavi',
       'Consumi/costi operativi', 'servizi esterni/costi operativi',
       'costo lavoro/costi operativi',
       'ammortam materiali/costi operativi', 'Val Agg/ITN', 'Ricavi/AN',
       'gg magazz', 'AC/PC', 'AC-mag/PC', 'Liq/PC', 'Liq/AN',
       'Cap Circ/AN', 'Patr netto/AN', 'Riserve+utile/AN',
       'Patr netto tang/AN', 'Patr netto/debiti totali',
       'Patr netto tan/Debiti tot+PN', 'Patr netto tan/Debiti tot-Liq+PN',
       'OF/RIC', 'OFN/RIC', 'OFN/EBITDA', 'OFN/EBIT', 'OFN/AN',
       'OFN/Autof lordo', 'Autof Lordo/AN (=cash flow/attivo)',
       'Auto Lordo-comp straord/AN', 'Deb totali/VA', 'Deb totali/Ric',
       'PC/ric', 'Deb finanziari (stimati)/VA',
       'Deb finanziari (stimati)/Ric', 'Debiti totali/EBITDA',
       'Debiti finanziari/EBITDA', 'Ebitda-servizio debito/AN', 'Ln(AN)',
       'Ln(RIC)']
```

Si inserisce X in un dataframe chiamato *data*:

```
data=pd.DataFrame(X, columns=cols_in)
data
```

	EBITDA/Ricavi	EBIT/ricavi	Utile corrente/ricavi	Risultato netto rettif/ricavi	Risultato netto rettif/AN	EBITDA/AN	ROA	ROE	ROE ante imposte	Val Agg Oper/Ricavi	Con
0	-3.44	-13.47	-8.65	-13.16	-6.82	-1.78	-2.65	-19.40	-20.94	11.48	
1	2.72	-5.56	1.43	0.93	0.64	1.86	2.08	1.82	2.11	15.32	
2	0.64	-5.86	-6.01	-0.58	-0.48	0.52	-3.32	-1.50	-18.92	10.96	
3	6.16	3.15	1.54	-0.33	-0.30	5.56	3.57	-1.00	1.17	13.91	
4	5.88	3.03	2.48	0.78	0.99	7.46	4.80	3.47	4.70	11.51	
...	...	...	...	...	...	...	...	...	...	...	
31831	33.26	24.37	0.00	0.00	0.00	3.23	3.23	-200.00	-200.00	33.33	
31832	33.26	24.37	0.00	0.00	0.00	12.50	12.50	-200.00	-200.00	33.33	
31833	33.26	24.37	0.00	0.00	0.00	11.11	11.11	0.00	0.00	33.33	
31834	33.26	0.00	-33.33	-33.33	-11.11	11.11	0.00	-12.50	-12.50	33.33	
31835	33.26	24.37	0.00	0.00	0.00	0.85	0.85	0.00	0.00	33.33	

31836 rows × 46 columns

Prima di procedere con l'estrazione dei fattori è necessario verificare la "*fattorializzazione*" del set di dati di cui si dispone; tale test di adeguatezza è composto dal:

- *Test di sfericità di Bartlett*: controlla attraverso il confronto tra una matrice di correlazione con la matrice identità se vi è una ridondanza tra le variabili che si possono riassumere con pochi fattori. L'ipotesi nulla del test è rappresentata dall'assunzione che le variabili siano ortogonali (non correlate); l'ipotesi alternativa è che le variabili non siano ortogonali, ovvero che siano sufficientemente correlate da far divergere significativamente la matrice di correlazione (R) da quella d'identità (I). Dunque:

$$H_0 : R = I \qquad \chi^2 = -\left[n - 1 - \frac{1}{6}(2p + 5)\right] ln\,|R|$$

dove n è il numero di osservazioni e p è il numero di variabili.

In codice Python:

```
from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity

chi_square_value, p_value= calculate_bartlett_sphericity(data)
chi_square_value, p_value
```

```
(2435868.941967734, 0.0)
```

dato che il valore del p-value è 0 (molto significativo), si rifiuta l'ipotesi nulla. Quindi, essendo la matrice di correlazione osservata significativamente diversa dalla matrice identità, si può utilizzare un'analisi fattoriale.

- *Test di Kaiser-Meyer-Olkin (KMO)*:

$$KMO = \frac{\sum_i \sum_j r_{ij}^2}{\sum_i \sum_j r_{ij}^2 + \sum_i \sum_j p_{ij}^2}$$

dove r_{ij} sono le correlazioni e p_{ij} sono le correlazioni parzializzate su tutte le altre. Quest'ultime indicano il valore di ogni correlazione dopo

aver rimosso il contributo di tutte le altre features non implicate; una parzializzata alta significa che le due variabili sono molto correlate fra loro, ma non hanno legami con nessun'altra all'interno del dataset.

Più le correlazioni parzializzate sono basse (KMO tende a 1), più il set sarà adatto a un'analisi fattoriale:

- ✓ se ≥ 0.9 è eccellente;
- ✓ fra 0.8 e 0.9 è buono;
- ✓ fra 0.7 e 0.8 è accettabile;
- ✓ fra 0.6 e 0.7 è mediocre;
- × se < 0.6 il dataset non è idoneo all'analisi fattoriale.

Nel caso pratico il KMO ha un valore di oltre 0.88, molto buono per questo tipo di analisi:

```
from factor_analyzer.factor_analyzer import calculate_kmo

kmo_all, kmo_model= calculate_kmo(data)
kmo_model
```

```
0.8832118089156898
```

Dunque, per entrambi i test il dataset è "fattorizzabile".

Si va a calcolare le correlazioni tra le features presenti all'interno del dataframe *data*:

```
corrm= data.corr()
corrm
```

	EBITDA/Ricavi	EBIT/ricavi	Utile corrente/ricavi	Risultato netto rettif/ricavi	Risultato netto rettif/AN	EBITDA/AN	ROA	ROE	ROE ante imposte
EBITDA/Ricavi	1.000000	0.934246	0.872333	0.789507	0.554818	0.653674	0.614920	0.467868	0.471486
EBIT/ricavi	0.934246	1.000000	0.942071	0.856768	0.564002	0.616030	0.628604	0.500212	0.508534
Utile corrente/ricavi	0.872333	0.942071	1.000000	0.906774	0.590461	0.605429	0.630646	0.527116	0.533833
Risultato netto rettif/ricavi	0.789507	0.856768	0.906774	1.000000	0.657354	0.559377	0.579023	0.554636	0.557535
Risultato netto rettif/AN	0.554818	0.564002	0.590461	0.657354	1.000000	0.809970	0.888596	0.642664	0.650660
EBITDA/AN	0.653674	0.616030	0.605429	0.559377	0.809970	1.000000	0.921851	0.538916	0.572854
ROA	0.614920	0.628604	0.630646	0.579023	0.888596	0.921851	1.000000	0.576080	0.607011
ROE	0.467868	0.500212	0.527116	0.554636	0.642664	0.538916	0.576080	1.000000	0.975761
ROE ante imposte	0.471486	0.508534	0.533833	0.557535	0.650660	0.572854	0.607011	0.975761	1.000000
Val Agg Oper/Ricavi	0.820318	0.766967	0.733541	0.665446	0.401973	0.515581	0.451978	0.358033	0.373443
Consumi/costi operativi	0.199480	0.302213	0.299484	0.288066	0.144319	0.096278	0.142286	0.217611	0.196693
servizi esterni/costi operativi	-0.336708	-0.400340	-0.418449	-0.400810	-0.148380	-0.164362	-0.141809	-0.274621	-0.262967
costo lavoro/costi operativi	0.125447	0.160695	0.186248	0.179185	0.017737	0.096136	0.029891	0.088618	0.114253
ammortam materiali/costi operativi	0.246764	0.059215	0.070188	0.059044	-0.007479	0.151402	-0.040434	0.056733	0.030190

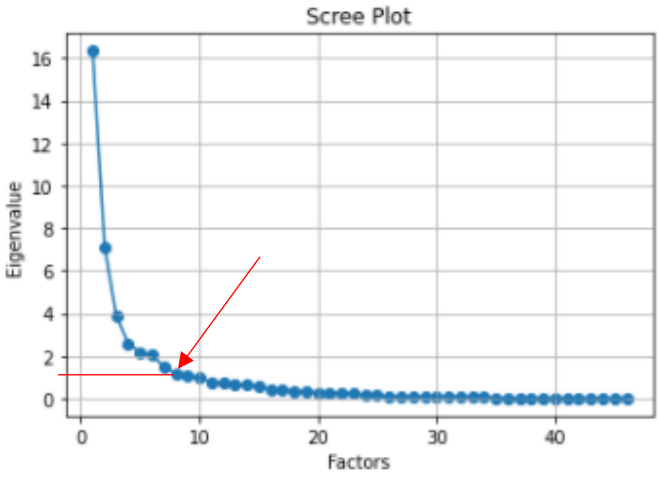
A questo punto si deve scegliere il numero di fattori da realizzare; per fare ciò viene utilizzato un grafico chiamato *Scree Plot*, basato sul calcolo degli autovalori:

```
eigen_values = np.linalg.eigvals(corrm)
eigen_values_cumvar = (eigen_values/46).cumsum()
eigen_values_cumvar
#autovalori
```

```
array([0.35519042, 0.51052608, 0.5947082 , 0.65083418, 0.6982213 ,
       0.74258703, 0.77505299, 0.80075928, 0.82404729, 0.84527363,
       0.86217817, 0.87771966, 0.89199211, 0.90573022, 0.91788114,
       0.92730331, 0.93587926, 0.94396795, 0.95051673, 0.95681559,
       0.96282658, 0.96806148, 0.97326309, 0.97626608, 0.97924065,
       0.9817937 , 0.98410467, 0.98607369, 0.987832  , 0.98938601,
       0.9907606 , 0.9920801 , 0.99329314, 0.99434723, 0.99523624,
       0.99544381, 0.9956747 , 0.99636904, 0.9966731 , 0.99704034,
       0.99741774, 0.99787468, 0.99836147, 0.99887012, 0.9994486 ,
       1.        ])
```

Gli autovalori rappresentano la varianza spiegata da un particolare fattore rispetto alla varianza totale; la figura mostra tale varianza *cumulata* man mano che aumenta il numero di autovalori: quindi, il primo autovalore spiega circa il 35% della varianza totale; se si aggiunge un secondo autovalore, questi, insieme, raggiungono il 51%. Il tutto fino ad arrivare al 100% della varianza piegata, corrispondente a un numero di autovalori uguale al numero di features.

```
plt.scatter(range(1,data.shape[1]+1),eigen_values)
plt.plot(range(1,data.shape[1]+1),eigen_values)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigenvalue')
plt.grid()
plt.show()
```



Il grafico Scree Plot ha come ascissa il numero di fattori e come ordinata il valore degli autovalori. In generale, si prende un numero *k* di fattori rispetto al quale l'autovalore corrispondente è circa 1; dopo varie prove, si è deciso di realizzare 8 fattori. Osservando i risultati del precedente comando si vede che i primi 8 fattori spiegano l'80% della varianza totale.

```
from factor_analyzer import FactorAnalyzer

#costruiamo 8 fattori
fa = FactorAnalyzer(n_factors=8, rotation='varimax', method='ml')
```

```
fa.fit(data)
```

```
FactorAnalyzer(method='ml', n_factors=8, rotation='varimax', rotation_kwargs={})
```

I parametri della funzione sono:

- *n_factors*: numero di fattori da realizzare.
- *rotation*: sono disponibili 3 principali metodi di rotazione: Quartimax, Promax e Varimax, che è quello che si è utilizzato; la *Varimax* (rotazione di tipo ortogonale) ruota la matrice dei fattori di carico (*factor loadings*, ovvero i coefficienti di correlazione tra le variabili e i fattori) in modo da massimizzare la somma delle varianze dei fattori di carico al quadrato, preservando al tempo stesso l'ortogonalità della matrice di carico; questo, generalmente, si traduce in fattori di carico elevati per un ristretto numero di variabili e bassi per il resto, rendendo il tutto più facile da interpretare.
- *method*: metodi standard per stimare il modello di analisi fattoriale; si è scelto la massima verosimiglianza (*ml*).

Si calcolano i factor loadings e si memorizzano in un nuovo dataframe 46x8 chiamato *ld*:

```
ld= fa.loadings_
ld=pd.DataFrame(ld, index=cols_in)
ld
```

	1	2	3	4	5	6	7	8
EBITDA/Ricavi	0.466961	0.083199	-0.227876	0.759349	-0.249706	0.117134	0.053814	0.011200
EBIT/ricavi	0.436944	0.063420	-0.366702	0.771949	-0.202034	-0.003232	0.048035	0.017124
Utile corrente/ricavi	0.450862	0.079221	-0.436647	0.700980	-0.178218	-0.003632	-0.103196	0.023330
Risultato netto rettif/ricavi	0.447094	0.077622	-0.423762	0.619040	-0.152651	-0.009896	-0.127629	0.010308
Risultato netto rettif/AN	0.847949	0.196221	-0.070572	0.187358	-0.096481	-0.052207	-0.136429	0.032186
EBITDA/AN	0.928870	0.073979	-0.142567	0.165424	-0.104619	0.079502	0.073978	0.080611
ROA	0.893166	0.140650	-0.074729	0.226521	-0.101248	-0.049311	0.038905	0.072217
ROE	0.516671	0.315771	-0.329023	0.127572	-0.209215	0.014295	-0.010901	-0.186930
ROE ante imposte	0.548393	0.240956	-0.334629	0.119283	-0.213646	0.021908	0.007679	-0.130142
Val Agg Oper/Ricavi	0.318053	0.033310	-0.222388	0.657939	-0.177679	0.605265	0.021372	0.109640
Consumi/costi operativi	0.068098	-0.046778	-0.415758	0.142506	-0.105135	-0.562681	0.008267	-0.190924
servizi esterni/costi operativi	-0.060638	0.068257	0.552940	-0.219120	0.136123	-0.009943	0.034944	0.127635
costo lavoro/costi operativi	-0.002720	-0.064747	-0.254002	0.079095	0.003351	0.803026	-0.033178	0.158216
ammortam materiali/costi operativi	0.081324	0.101835	0.018189	0.060240	-0.050727	0.399403	-0.018742	-0.135550
Val Agg/ITN	0.165765	-0.017549	-0.117700	0.093228	0.000977	0.082410	-0.035196	0.295409
Ricavi/AN	0.329862	-0.218018	-0.588719	-0.036555	-0.001068	-0.136130	0.009838	0.113777
gg magazz	-0.231785	-0.038422	-0.084720	0.047248	-0.047566	0.007379	-0.002852	-0.163708
AC/PC	-0.015403	0.858826	0.064655	-0.019764	-0.050052	-0.037749	-0.019638	0.422621
AC-mag/PC	0.022985	0.850886	0.101666	-0.019223	-0.035055	-0.017075	-0.015624	0.478654
Liq/PC	0.040155	0.724252	0.135287	-0.007563	-0.017305	-0.031032	-0.036388	0.387374
Liq/AN	0.138049	0.425384	0.123971	-0.029953	0.036336	-0.060916	-0.104544	0.306182
Cap Circ/AN	0.160621	0.744311	-0.219125	0.023040	-0.054769	-0.020275	-0.042733	0.216424
Patr netto/AN	0.175622	0.905808	-0.102572	0.054197	-0.008966	0.085683	-0.051073	-0.291880
Riserve+utile/AN	0.328917	0.638873	-0.285392	0.112697	-0.060932	0.101900	-0.143618	-0.184930
Patr netto tang/AN	0.177966	0.902646	-0.148483	0.057326	-0.015716	0.077821	-0.069408	-0.326857
Patr netto/debiti totali	0.000376	0.882624	0.078331	0.006701	-0.009164	0.063891	-0.023721	-0.028335
Patr netto tan/Debiti tot+PN	0.167016	0.894295	-0.128234	0.050166	-0.012746	0.098025	-0.072279	-0.290051
Patr netto tan/Debiti tot-Liq+PN	0.168062	0.882407	-0.053989	0.041879	-0.002481	0.073624	-0.084650	-0.145318
OF/RIC	-0.166880	-0.017628	0.713633	-0.133657	0.086223	-0.084243	0.548497	-0.070297
OFN/RIC	-0.189821	-0.095917	0.613986	-0.065766	0.054984	-0.046875	0.681887	-0.078683
OFN/EBITDA	-0.590258	-0.051662	0.253469	-0.286498	0.678930	-0.039305	0.092738	0.039046
OFN/EBIT	-0.629638	-0.071110	0.215793	-0.249190	0.406710	0.049410	0.090897	-0.022080
OFN/AN	-0.124818	-0.382395	-0.011443	0.045249	-0.005483	-0.035060	0.689286	0.040624
OFN/Autof lordo	-0.589245	-0.087724	0.278856	-0.173452	0.445477	-0.087908	0.286251	0.028393
Autof Lordo/AN (=cash flow/attivo)	0.898871	0.106366	-0.160760	0.102251	-0.068666	0.151390	-0.112371	0.070511
Auto Lordo-comp straord/AN	0.911348	0.104285	-0.173094	0.137949	-0.107942	0.151621	-0.096226	0.066522
Deb totali/VA	-0.325477	-0.260786	0.315183	0.310303	-0.031666	-0.340460	-0.003387	-0.069337
Deb totali/Ric	-0.141753	-0.182419	0.874196	-0.205771	0.094114	-0.121318	-0.168658	-0.003560
PC/ric	-0.146943	-0.223480	0.844395	-0.205082	0.108920	-0.094059	-0.167333	-0.107042
Deb finanziari (stimati)/VA	-0.305741	-0.144644	0.418964	0.230935	-0.008407	-0.285992	0.266471	-0.058345
Deb finanziari (stimati)/Ric	-0.153248	-0.049586	0.823877	-0.157389	0.077520	-0.119128	0.186943	-0.017733
Debiti totali/EBITDA	-0.556431	-0.040348	0.285458	-0.283919	0.703620	-0.039902	-0.052154	0.053856
Debiti finanziari/EBITDA	-0.550707	-0.018262	0.228646	-0.326463	0.726882	-0.021689	-0.002768	0.048693
Ebitda-servizio debito/AN	0.795169	0.242439	-0.249776	0.105545	-0.094311	0.117829	-0.219896	-0.003015
Ln(AN)	0.025833	0.060571	-0.298639	0.021619	-0.070256	-0.070970	-0.215859	-0.306344
Ln(RIC)	0.153072	-0.036779	-0.680072	0.109897	-0.101505	-0.030666	-0.158984	-0.180622

A questo punto, per ogni fattore si va a vedere quali variabili hanno un load factor alto, quindi con quali features è maggiormente correlato, in modo tale da individuare quale categoria il fattore spiega; infatti, per entrare nel modello è necessario che ogni fattore abbia una sua logica razionale (ovvero che rappresenti qualcosa di concreto):

FEATURES	1	FEATURES	1
EBITDA/Ricavi	0,466960753	Patr netto/AN	0,175622349
EBIT/ricavi	0,436943937	Riserve+utile/AN	0,328917404
Utile corrente/ricavi	0,450862104	Patr netto tang/AN	0,177966195
Risultato netto rettif/ricavi	0,447094416	Patr netto/debiti totali	0,000375631
Val Agg Oper/Ricavi	0,318052678	Patr netto tan/Debiti tot+PN	0,167015642
Risultato netto rettif/AN	0,847949004	Patr netto tan/Debiti tot-Liq+PN	0,168062191
EBITDA/AN	0,92887031	OF/RIC	-0,166879627
Ebitda-servizio debito/AN	0,795169452	OFN/RIC	-0,189820797
ROA	0,893166295	OFN/EBITDA	-0,59025799
ROE	0,516671147	OFN/EBIT	-0,629637793
ROE ante imposte	0,548392841	OFN/AN	-0,124818286
Consumi/costi operativi	0,068097992	OFN/Autof lordo	-0,589245067
servizi esterni/costi operativi	-0,060637562	Autof Lordo/AN (=cash flow/attivo)	0,898871056
costo lavoro/costi operativi	-0,002720168	Auto Lordo-comp straord/AN	0,911348154
ammortam materiali/costi operativi	0,081324398	Deb totali/VA	-0,325477262
Val Agg/ITN	0,165764778	Deb totali/Ric	-0,141752898
Ricavi/AN	0,329862315	PC/ric	-0,146943074
gg magazz	-0,231784822	Deb finanziari (stimati)/VA	-0,305740916
AC/PC	-0,015402658	Deb finanziari (stimati)/Ric	-0,153248288
AC-mag/PC	0,022985214	Debiti totali/EBITDA	-0,556431351
Liq/PC	0,040154642	Debiti finanziari/EBITDA	-0,550706778
Liq/AN	0,13804926	Ln(AN)	0,025833106
Cap Circ/AN	0,160620515	Ln(RIC)	0,153071572

Vengono selezionate le variabili:

- Risultato netto rettif/AN;
- EBITDA/AN;
- Ebitda-servizio debito/AN;
- ROA;
- ROE;
- ROE ante imposte;
- Autof Lordo/AN (=cash flow/attivo);

- Auto Lordo-comp straord/AN.

Dunque, il primo fattore rappresenta la categoria "*Redditività sul capitale investito e Capacità di autofinanziamento*" e, per le ragioni spiegate in precedenza, dovrà necessariamente entrare nel modello con un coefficiente negativo (-).

FEATURES	2	FEATURES	2
EBITDA/Ricavi	0,083198998	Patr netto/AN	0,905807922
EBIT/ricavi	0,063420338	Riserve+utile/AN	0,638872521
Utile corrente/ricavi	0,079221425	Patr netto tang/AN	0,902645937
Risultato netto rettif/ricavi	0,077621736	Patr netto/debiti totali	0,882623857
Val Agg Oper/Ricavi	0,033310345	Patr netto tan/Debiti tot+PN	0,894294622
Risultato netto rettif/AN	0,196220818	Patr netto tan/Debiti tot-Liq+PN	0,882407298
EBITDA/AN	0,073978779	OF/RIC	-0,017627895
Ebitda-servizio debito/AN	0,242438975	OFN/RIC	-0,095917107
ROA	0,140650112	OFN/EBITDA	-0,051662274
ROE	0,315770863	OFN/EBIT	-0,071110261
ROE ante imposte	0,240955978	OFN/AN	-0,382395439
Consumi/costi operativi	-0,046777861	OFN/Autof lordo	-0,087724043
servizi esterni/costi operativi	0,068256519	Autof Lordo/AN (=cash flow/attivo)	0,106366414
costo lavoro/costi operativi	-0,064746954	Auto Lordo-comp straord/AN	0,104285123
ammortam materiali/costi operativi	0,101835004	Deb totali/VA	-0,260786078
Val Agg/ITN	-0,017548545	Deb totali/Ric	-0,182419102
Ricavi/AN	-0,218017894	PC/ric	-0,223480108
gg magazz	-0,038422426	Deb finanziari (stimati)/VA	-0,144644428
AC/PC	0,858825822	Deb finanziari (stimati)/Ric	-0,049585963
AC-mag/PC	0,850886218	Debiti totali/EBITDA	-0,04034808
Liq/PC	0,724252145	Debiti finanziari/EBITDA	-0,01826209
Liq/AN	0,425384413	Ln(AN)	0,060571255
Cap Circ/AN	0,744310953	Ln(RIC)	-0,036778663

Il secondo fattore risulta spiegare la "*Liquidità e Struttura finanziaria*" essendo fortemente correlato con:

- AC/PC;
- AC-mag/PC;
- Liq/PC;
- Liq/AN;
- Cap Circ/AN;
- Patr netto/AN;

- Riserve+utile/AN;
- Patr netto tang/AN;
- Patr netto/debiti totali;
- Patr netto tan/Debiti tot+PN;
- Patr netto tan/Debiti tot-liq+PN.

Anche questo fattore deve avere coefficiente negativo (-).

Si ripete tale analisi per ognuno dei fattori, ottenendo il seguente risultato:

- Fattore 3: "*Peso del debito sui ricavi*" (+):

FEATURES	3	FEATURES	3
EBITDA/Ricavi	-0,227876089	Patr netto/AN	-0,102572322
EBIT/ricavi	-0,366701603	Riserve+utile/AN	-0,285391867
Utile corrente/ricavi	-0,436647193	Patr netto tang/AN	-0,148483169
Risultato netto rettif/ricavi	-0,423761573	Patr netto/debiti totali	0,078331226
Val Agg Oper/Ricavi	-0,222388276	Patr netto tan/Debiti tot+PN	-0,128234452
Risultato netto rettif/AN	-0,070572045	Patr netto tan/Debiti tot-Liq+PN	-0,053988736
EBITDA/AN	-0,142566632	OF/RIC	0,713633301
Ebitda-servizio debito/AN	-0,249776217	OFN/RIC	0,613986058
ROA	-0,074728548	OFN/EBITDA	0,253469345
ROE	-0,329022543	OFN/EBIT	0,215792634
ROE ante imposte	-0,334629133	OFN/AN	-0,011443144
Consumi/costi operativi	-0,41575801	OFN/Autof lordo	0,278856059
servizi esterni/costi operativi	0,552940331	Autof Lordo/AN (=cash flow/attivo)	-0,160760196
costo lavoro/costi operativi	-0,254002076	Auto Lordo-comp straord/AN	-0,173093858
ammortam materiali/costi operativi	0,018189105	Deb totali/VA	0,315183091
Val Agg/ITN	-0,117700376	Deb totali/Ric	0,874196067
Ricavi/AN	-0,588718628	PC/ric	0,844395061
gg magazz	-0,084719914	Deb finanziari (stimati)/VA	0,41896392
AC/PC	0,064655423	Deb finanziari (stimati)/Ric	0,823877337
AC-mag/PC	0,101665974	Debiti totali/EBITDA	0,285457772
Liq/PC	0,13528667	Debiti finanziari/EBITDA	0,228645869
Liq/AN	0,123970853	Ln(AN)	-0,298639041
Cap Circ/AN	-0,219125066	Ln(RIC)	-0,680071967

- Fattore 4: "*Redditività sui ricavi*" (-):

FEATURES	4	FEATURES	4
EBITDA/Ricavi	0,759348986	Patr netto/AN	0,054196916
EBIT/ricavi	0,771948944	Riserve+utile/AN	0,112696528
Utile corrente/ricavi	0,700980194	Patr netto tang/AN	0,057325537
Risultato netto rettif/ricavi	0,619040109	Patr netto/debiti totali	0,006701241
Val Agg Oper/Ricavi	0,657939163	Patr netto tan/Debiti tot+PN	0,050166064
Risultato netto rettif/AN	0,187358145	Patr netto tan/Debiti tot-Liq+PN	0,04187947
EBITDA/AN	0,165423627	OF/RIC	-0,133656955
Ebitda-servizio debito/AN	0,105544746	OFN/RIC	-0,065765605
ROA	0,22652121	OFN/EBITDA	-0,286498156
ROE	0,127572115	OFN/EBIT	-0,249190333
ROE ante imposte	0,119282534	OFN/AN	0,045249092
Consumi/costi operativi	0,142506419	OFN/Autof lordo	-0,173451977
servizi esterni/costi operativi	-0,219119599	Autof Lordo/AN (=cash flow/attivo)	0,102251216
costo lavoro/costi operativi	0,079095242	Auto Lordo-comp straord/AN	0,137948889
ammortam materiali/costi operativi	0,060239843	Deb totali/VA	0,310302554
Val Agg/ITN	0,093227613	Deb totali/Ric	-0,205771126
Ricavi/AN	-0,036554771	PC/ric	-0,205081688
gg magazz	0,047247606	Deb finanziari (stimati)/VA	0,230935475
AC/PC	-0,019764303	Deb finanziari (stimati)/Ric	-0,15738889
AC-mag/PC	-0,019222691	Debiti totali/EBITDA	-0,283919296
Liq/PC	-0,007563109	Debiti finanziari/EBITDA	-0,326463496
Liq/AN	-0,029953085	Ln(AN)	0,021619314
Cap Circ/AN	0,023040216	Ln(RIC)	0,109896729

- Fattore 5: "*Capacità di rimborso del debito*" (+):

FEATURES	5	FEATURES	5
EBITDA/Ricavi	-0,249706165	Patr netto/AN	-0,008966489
EBIT/ricavi	-0,202034374	Riserve+utile/AN	-0,060931835
Utile corrente/ricavi	-0,178218362	Patr netto tang/AN	-0,015715731
Risultato netto rettif/ricavi	-0,152651467	Patr netto/debiti totali	-0,00916442
Val Agg Oper/Ricavi	-0,177678662	Patr netto tan/Debiti tot+PN	-0,012745726
Risultato netto rettif/AN	-0,096481135	Patr netto tan/Debiti tot-Liq+PN	-0,002480508
EBITDA/AN	-0,104618748	OF/RIC	0,086223129
Ebitda-servizio debito/AN	-0,094310809	OFN/RIC	0,054984284
ROA	-0,101248357	OFN/EBITDA	0,678930454
ROE	-0,209214579	OFN/EBIT	0,40670961
ROE ante imposte	-0,213646162	OFN/AN	-0,005482775
Consumi/costi operativi	-0,105134824	OFN/Autof lordo	0,445476923
servizi esterni/costi operativi	0,136123134	Autof Lordo/AN (=cash flow/attivo)	-0,068666391
costo lavoro/costi operativi	0,003351261	Auto Lordo-comp straord/AN	-0,107941772
ammortam materiali/costi operativi	-0,050727202	Deb totali/VA	-0,031665653
Val Agg/ITN	0,000976933	Deb totali/Ric	0,09411366
Ricavi/AN	-0,001068193	PC/ric	0,108919684
gg magazz	-0,047566046	Deb finanziari (stimati)/VA	-0,008406908
AC/PC	-0,050052063	Deb finanziari (stimati)/Ric	0,077520264
AC-mag/PC	-0,035055104	Debiti totali/EBITDA	0,703619828
Liq/PC	-0,017304564	Debiti finanziari/EBITDA	0,726882056
Liq/AN	0,03633629	Ln(AN)	-0,070256444
Cap Circ/AN	-0,054768839	Ln(RIC)	-0,101505233

- Fattore 6: "*Incidenza dei costi fissi sui costi operativi*" (+):

FEATURES	6	FEATURES	6
EBITDA/Ricavi	0,117134264	Patr netto/AN	0,085682565
EBIT/ricavi	-0,003231584	Riserve+utile/AN	0,101899583
Utile corrente/ricavi	-0,003632222	Patr netto tang/AN	0,077820788
Risultato netto rettif/ricavi	-0,009896016	Patr netto/debiti totali	0,063891095
Val Agg Oper/Ricavi	0,605264733	Patr netto tan/Debiti tot+PN	0,098025148
Risultato netto rettif/AN	-0,052206911	Patr netto tan/Debiti tot-Liq+PN	0,073623702
EBITDA/AN	0,079502314	OF/RIC	-0,084242521
Ebitda-servizio debito/AN	0,117829056	OFN/RIC	-0,046874751
ROA	-0,049310731	OFN/EBITDA	-0,039305412
ROE	0,014295488	OFN/EBIT	0,049410064
ROE ante imposte	0,021908284	OFN/AN	-0,035060271
Consumi/costi operativi	-0,562680626	OFN/Autof lordo	-0,087908402
servizi esterni/costi operativi	-0,009942997	Autof Lordo/AN (=cash flow/attivo)	0,151390429
costo lavoro/costi operativi	0,80302587	Auto Lordo-comp straord/AN	0,151621437
ammortam materiali/costi operativi	0,399403252	Deb totali/VA	-0,340459618
Val Agg/ITN	0,08240983	Deb totali/Ric	-0,121318421
Ricavi/AN	-0,136130353	PC/ric	-0,094059019
gg magazz	0,007378701	Deb finanziari (stimati)/VA	-0,28599199
AC/PC	-0,037749024	Deb finanziari (stimati)/Ric	-0,119127751
AC-mag/PC	-0,0170746	Debiti totali/EBITDA	-0,039901923
Liq/PC	-0,031031666	Debiti finanziari/EBITDA	-0,021688713
Liq/AN	-0,060916199	Ln(AN)	-0,070970131
Cap Circ/AN	-0,02027497	Ln(RIC)	-0,030666207

- Fattore 7: "*Peso degli oneri finanziari (netti)*" (+):

FEATURES	7	FEATURES	7
EBITDA/Ricavi	0,053813631	Patr netto/AN	-0,051072979
EBIT/ricavi	0,048034773	Riserve+utile/AN	-0,143618118
Utile corrente/ricavi	-0,103195758	Patr netto tang/AN	-0,069408025
Risultato netto rettif/ricavi	-0,127629489	Patr netto/debiti totali	-0,023721478
Val Agg Oper/Ricavi	0,021372109	Patr netto tan/Debiti tot+PN	-0,072278595
Risultato netto rettif/AN	-0,136429216	Patr netto tan/Debiti tot-Liq+PN	-0,084649591
EBITDA/AN	0,073978412	OF/RIC	0,548497482
Ebitda-servizio debito/AN	-0,219896267	OFN/RIC	0,681886823
ROA	0,038905192	OFN/EBITDA	0,092738047
ROE	-0,010900908	OFN/EBIT	0,09089669
ROE ante imposte	0,007678731	OFN/AN	0,689286271
Consumi/costi operativi	0,008266558	OFN/Autof lordo	0,286250679
servizi esterni/costi operativi	0,03494385	Autof Lordo/AN (=cash flow/attivo)	-0,112371378
costo lavoro/costi operativi	-0,033178114	Auto Lordo-comp straord/AN	-0,096225644
ammortam materiali/costi operativi	-0,018741831	Deb totali/VA	-0,003386528
Val Agg/ITN	-0,035196444	Deb totali/Ric	-0,168657531
Ricavi/AN	0,009837823	PC/ric	-0,167332837
gg magazz	-0,002851623	Deb finanziari (stimati)/VA	0,266471067
AC/PC	-0,019638217	Deb finanziari (stimati)/Ric	0,186943284
AC-mag/PC	-0,015624275	Debiti totali/EBITDA	-0,052154327
Liq/PC	-0,036388395	Debiti finanziari/EBITDA	-0,002768263
Liq/AN	-0,104543589	Ln(AN)	-0,215859398
Cap Circ/AN	-0,042732921	Ln(RIC)	-0,158983787

- Fattore 8: "*Dimensione*" (-):

FEATURES	8	FEATURES	8
EBITDA/Ricavi	0,011200277	Patr netto/AN	-0,291880198
EBIT/ricavi	0,017124227	Riserve+utile/AN	-0,184930038
Utile corrente/ricavi	0,023329801	Patr netto tang/AN	-0,32685739
Risultato netto rettif/ricavi	0,010308416	Patr netto/debiti totali	-0,028335479
Val Agg Oper/Ricavi	0,109640436	Patr netto tan/Debiti tot+PN	-0,290051463
Risultato netto rettif/AN	0,032185575	Patr netto tan/Debiti tot-Liq+PN	-0,145318136
EBITDA/AN	0,080611238	OF/RIC	-0,0702974
Ebitda-servizio debito/AN	-0,003015345	OFN/RIC	-0,078682601
ROA	0,072216513	OFN/EBITDA	0,039045867
ROE	-0,186929778	OFN/EBIT	-0,022079983
ROE ante imposte	-0,130142041	OFN/AN	0,040624182
Consumi/costi operativi	-0,190923995	OFN/Autof lordo	0,028392724
servizi esterni/costi operativi	0,12763535	Autof Lordo/AN (=cash flow/attivo)	0,070510801
costo lavoro/costi operativi	0,158216305	Auto Lordo-comp straord/AN	0,066522295
ammortam materiali/costi operativi	-0,135550058	Deb totali/VA	-0,069336598
Val Agg/ITN	0,295408513	Deb totali/Ric	-0,003559903
Ricavi/AN	0,113777018	PC/ric	-0,107042408
gg magazz	-0,163708387	Deb finanziari (stimati)/VA	-0,058345036
AC/PC	0,422620818	Deb finanziari (stimati)/Ric	-0,017732697
AC-mag/PC	0,478653871	Debiti totali/EBITDA	0,053856247
Liq/PC	0,387373955	Debiti finanziari/EBITDA	0,04869342
Liq/AN	0,306181643	Ln(AN)	-0,306343562
Cap Circ/AN	0,216424092	Ln(RIC)	-0,180621557

Successivamente, si va a costruire il nuovo dataset moltiplicando la matrice *data* (31836x46) per quella dei factor loadings *ld* (46x8); la risultante, contenente 31836 osservazioni e 8 variabili (fattori), è stata memorizzata nel dataframe X:

```
#creiamo array numpy

X= data.values
ld= ld.values
```

```
data.shape
```

```
(31836, 46)
```

```
ld.shape
```

```
(46, 8)
```

```
X=X.dot(ld) #prodotto tra matrici
```

```
cols=['F1','F2','F3','F4','F5','F6','F7','F8']

X= pd.DataFrame(X, columns=cols)
X
```

	F1	F2	F3	F4	F5	F6	F7	F8
0	-2018.135776	-89.851876	1226.232652	-540.040705	1843.150440	-439.434362	33.152268	18.181339
1	-458.305806	182.194206	368.173640	78.522557	158.276201	-200.806626	17.090908	-36.775821
2	-619.460083	103.475428	454.985033	53.502587	280.983335	-258.221130	19.842617	-31.154059
3	-316.318609	136.844391	365.758550	196.323594	38.964480	-276.143021	41.186747	-19.933704
4	-195.412634	146.269148	255.095249	198.294481	-10.047755	-237.419107	17.099149	-17.470253
...	...	...	...	...	...	...	...	...
31831	-2382.951812	-1810.479229	5965.200323	-413.939291	923.218759	-1602.446328	5.046828	-202.533574
31832	-1604.003497	-930.368347	2685.448723	114.735997	510.278622	-941.701823	592.270261	-38.695339
31833	-823.034774	1688.548290	1849.974703	12.816183	333.629135	-519.404734	605.686884	455.372048
31834	-1077.513628	1650.576934	1949.102244	-127.028692	470.617233	-501.594085	638.181848	449.746742
31835	-829.275589	1659.375184	1835.605515	13.004462	330.379660	-515.761004	601.886176	446.323393

31836 rows × 8 columns

```
X = X.loc[:, X.columns]
y = credit.loc[:, credit.columns == 'FLAG di sana in liquid + anomala']
y #y_anno
```

	FLAG di sana in liquid + anomala
0	0
1	0
2	0
3	0
4	0
...	...
31831	0
31832	0
31833	0
31834	0
31835	0

31836 rows × 1 columns

Gli 8 fattori rappresentano il dataset iniziale; si procede con la fase di features/factors selection calcolando i coefficienti di correlazione tra ogni fattore e la target *y anno*:

```
#COEFF DI CORRELAZIONE TRA OGNI FATTORE E LA y:
for i in range(len(cols)):
    print(np.corrcoef(X[cols[i]],y['FLAG di sana in liquid + anomala'])[0][1], cols[i])
```

```
-0.11926842429128463 F1
-0.08457419485642785 F2
0.10071552842194795 F3
-0.10630910320806918 F4
0.11611240568268341 F5
-0.08123791045418705 F6
0.015325713800046915 F7
0.012258752846679015 F8
```

Tutti le correlazioni presentano segno corretto tranne per i fattori F6 e F8; per questo motivo vengono a priori rimossi dal modello:

```
elim= ['F6','F8']

X= X.drop(elim, axis=1)
X
```

	F1	F2	F3	F4	F5	F7
0	-2018.135776	-89.851876	1226.232652	-540.040705	1843.150440	33.152268
1	-458.305806	182.194206	368.173640	78.522557	158.276201	17.090908
2	-619.460083	103.475428	454.985033	53.502587	280.983335	19.842617
3	-316.318609	136.844391	365.758550	196.323594	38.964480	41.186747
4	-195.412634	146.269148	255.095249	198.294481	-10.047755	17.099149
...	...	...	...	...	...	...
31831	-2382.951812	-1810.479229	5965.200323	-413.939291	923.218759	5.046828
31832	-1604.003497	-930.368347	2685.448723	114.735997	510.278622	592.270261
31833	-823.034774	1688.548290	1849.974703	12.816183	333.629135	605.686884
31834	-1077.513628	1650.576934	1949.102244	-127.028692	470.617233	638.181848
31835	-829.275589	1659.375184	1835.605515	13.004462	330.379660	601.886176

31836 rows × 6 columns

La 1° versione di questo modello ripercorre la stessa procedura dei due precedenti: vengono inseriti nel modello tutti i fattori simultaneamente e, ad ogni step, si rimuove quel fattore (uno alla volta) che presenta coefficiente errato e il più alto p-value:

```
import statsmodels.api as sm
logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
==========================================================================
Model:              Logit                            Pseudo R-squared: -0.547
Dependent Variable: FLAG di sana in liquid + anomala AIC:              3125.5229
Date:               2020-11-13 17:33                 BIC:              3175.7330
No. Observations:   31836                            Log-Likelihood:   -1556.8
Df Model:           5                                LL-Null:          -1006.5
Df Residuals:       31830                            LLR p-value:      1.0000
Converged:          1.0000                           Scale:            1.0000
No. Iterations:     10.0000
--------------------------------------------------------------------------------
             Coef.       Std.Err.         z          P>|z|        [0.025       0.975]
--------------------------------------------------------------------------------
F1           0.0012        0.0006        2.1345       0.0328       0.0001       0.0024
F2          -0.0083        0.0004      -22.9244       0.0000      -0.0091      -0.0076
F3          -0.0040        0.0002      -25.6600       0.0000      -0.0044      -0.0037
F4          -0.0146        0.0005      -32.3550       0.0000      -0.0155      -0.0138
F5          -0.0062        0.0006       -9.8911       0.0000      -0.0074      -0.0049
F7           0.0046        0.0009        5.2666       0.0000       0.0029       0.0064
==========================================================================
```

Il primo fattore da eliminare risulta F1; dopo di che, come già esposto, viene stimato nuovamente il modello (con un fattore in meno) e ripetuto iterativamente il processo fin quando non si raggiunge la situazione in cui si hanno tutti i fattori con coefficienti corretti e p-value < 0.05:

```
==========================================================================
Model:              Logit                            Pseudo R-squared: -18.466
Dependent Variable: FLAG di sana in liquid + anomala AIC:              39187.3786
Date:               2020-11-13 17:37                 BIC:              39195.7470
No. Observations:   31836                            Log-Likelihood:   -19593.
Df Model:           0                                LL-Null:          -1006.5
Df Residuals:       31835                            LLR p-value:      nan
Converged:          1.0000                           Scale:            1.0000
No. Iterations:     5.0000
--------------------------------------------------------------------------------
          Coef.        Std.Err.          z          P>|z|        [0.025       0.975]
--------------------------------------------------------------------------------
F2       -0.0013         0.0000       -60.3746       0.0000       -0.0014       -0.0013
==========================================================================
```

Il modello in questa 1° versione contiene esclusivamente il fattore F2:

```
X
```

	F2
0	-89.851876
1	182.194206
2	103.475428
3	136.844391
4	146.269148
...	...
31831	-1810.479229
31832	-930.368347
31833	1688.548290
31834	1650.576934
31835	1659.375184

31836 rows × 1 columns

Si procede con lo split e normalizzazione; questo e i prossimi modelli di regressione logistica con analisi fattoriale hanno per oggetto la medesima fase di ottimizzazione (*solver* e *penalty*) e convalida incrociata, con i seguenti risultati:

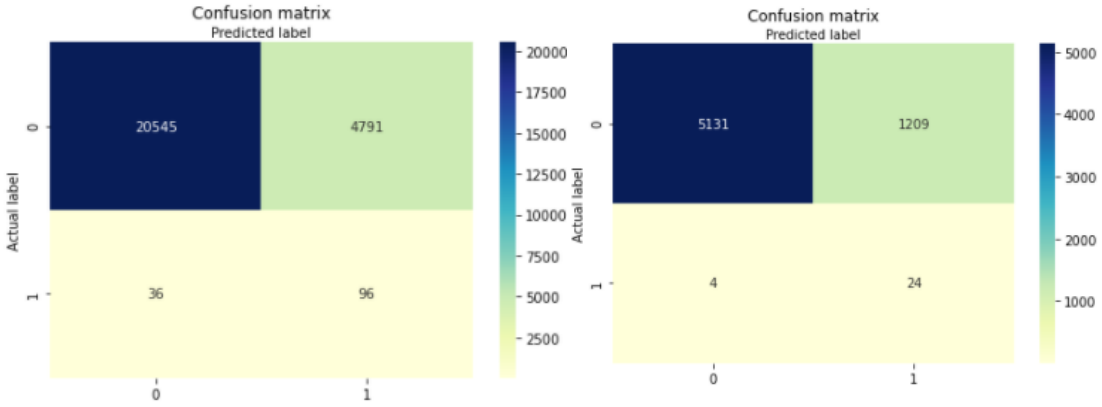
```
Best: 0.727273 using {'solver': 'lbfgs', 'penalty': 'none'}

Fold 1 score=0.8049
Fold 2 score=0.8057
Fold 3 score=0.8229
Fold 4 score=0.7994
Fold 5 score=0.8190
Fold 6 score=0.8186
Fold 7 score=0.8194
Fold 8 score=0.8127
Fold 9 score=0.7883
Fold 10 score=0.8052

Validation Accuracy = 0.81
```

Dopo la convalida, si ripete l'addestramento sull'intero set di training:

```
logisticRegr.fit(X_train, y_train)
```

```
LogisticRegression(class_weight='balanced', multi_class='ovr',
                   random_state=123456789)
```

Si effettua la predizione prima sul training e, successivamente, sul test set:




```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.81	0.90	25336
1	0.02	0.73	0.04	132
accuracy			0.81	25468
macro avg	0.51	0.77	0.47	25468
weighted avg	0.99	0.81	0.89	25468

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.81	0.89	6340
1	0.02	0.86	0.04	28
accuracy			0.81	6368
macro avg	0.51	0.83	0.47	6368
weighted avg	0.99	0.81	0.89	6368

Utilizzando la target *anno* si riscontra nuovamente, tra le due classi, un forte sbilanciamento su precisione e f1; al contrario, il richiamo è soddisfacente (in particolare se si pensa che il modello è composto solamente da un fattore) e in aumento, dal train al test set sulla classe delle anomale, di 13 punti percentuali senza intaccare le performance sulla classe delle sane e quelle sull'accuracy.

[2° versione] (target *anno*)

Il modello appena descritto, pur offrendo discreti risultati, non convince per il fatto che contenga unicamente il fattore F2; dunque, se ne è sviluppata una 2° versione: invece di costruire il modello partendo da tutte le variabili, queste si *inseriscono una alla volta* per osservare immediatamente con quale segno entrano nel modello, evitando in questo modo di rimuovere a priori quelle con correlazione errata; il primo fattore inserito sarà quello con la più alta correlazione con la target *anno*:

```
#COEFF DI CORRELAZIONE TRA OGNI FATTORE E LA y:
for i in range(len(cols)):
    print(np.corrcoef(X[cols[i]],y['FLAG di sana in liquid + anomala'])[0][1], cols[i])
```

```
-0.11926842429128463 F1
-0.08457419485642785 F2
0.10071552842194795 F3
-0.10630910320806918 F4
0.11611240568268341 F5
-0.08123791045418705 F6
0.015325713800046915 F7
0.012258752846679015 F8
```

F1 è il primo fattore a essere inserito. Successivamente, si verifica il segno del coefficiente con il quale entra nel modello (se corretto viene mantenuto, altrimenti viene rimosso) e si procede con l'aggiunta del fattore successivo con la maggiore correlazione tra quelli residui:

```
import statsmodels.api as sm
logit_model=sm.Logit(y,X['F1'])
result=logit_model.fit()
print(result.summary2())
```

```
                                 Results: Logit
=================================================================================
Model:              Logit                            Pseudo R-squared: -16.066
Dependent Variable: FLAG di sana in liquid + anomala AIC:              34355.5706
Date:               2020-11-13 22:54                 BIC:              34363.9389
No. Observations:   31836                            Log-Likelihood:   -17177.
Df Model:           0                                LL-Null:          -1006.5
Df Residuals:       31835                            LLR p-value:      nan
Converged:          1.0000                           Scale:            1.0000
No. Iterations:     7.0000
---------------------------------------------------------------------------------
            Coef.        Std.Err.          z          P>|z|        [0.025       0.975]
---------------------------------------------------------------------------------
F1          0.0019         0.0000       56.7418       0.0000       0.0018       0.0020
=================================================================================
```

Il coefficiente ha segno errato (dovrebbe essere negativo), quindi F1 viene rimosso. Il prossimo fattore è F5:

```
logit_model=sm.Logit(y,X[['F5']])
result=logit_model.fit()
print(result.summary2())
```

```
                          Results: Logit
==========================================================================
Model:              Logit                          Pseudo R-squared: -16.887
Dependent Variable: FLAG di sana in liquid + anomala AIC:              36008.3235
Date:               2020-11-13 22:54               BIC:              36016.6919
No. Observations:   31836                          Log-Likelihood:   -18003.
Df Model:           0                              LL-Null:          -1006.5
Df Residuals:       31835                          LLR p-value:      nan
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     8.0000
--------------------------------------------------------------------------------
          Coef.        Std.Err.          z          P>|z|         [0.025        0.975]
--------------------------------------------------------------------------------
F5       -0.0027         0.0001       -36.9120       0.0000       -0.0028       -0.0025
==========================================================================
```

```
logit_model=sm.Logit(y,X[['F4']])
result=logit_model.fit()
print(result.summary2())
```

```
                          Results: Logit
==========================================================================
Model:              Logit                          Pseudo R-squared: -20.786
Dependent Variable: FLAG di sana in liquid + anomala AIC:              43858.3747
Date:               2020-11-13 22:54               BIC:              43866.7431
No. Observations:   31836                          Log-Likelihood:   -21928.
Df Model:           0                              LL-Null:          -1006.5
Df Residuals:       31835                          LLR p-value:      nan
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     4.0000
--------------------------------------------------------------------------------
             Coef.        Std.Err.          z          P>|z|         [0.025        0.975]
--------------------------------------------------------------------------------
F4           0.0004         0.0000       16.3380       0.0000        0.0003        0.0004
==========================================================================
```

```
logit_model=sm.Logit(y,X[['F3']])
result=logit_model.fit()
print(result.summary2())
```

```
                          Results: Logit
==========================================================================
Model:              Logit                          Pseudo R-squared: -10.593
Dependent Variable: FLAG di sana in liquid + anomala AIC:              23339.7495
Date:               2020-11-13 22:54               BIC:              23348.1179
No. Observations:   31836                          Log-Likelihood:   -11669.
Df Model:           0                              LL-Null:          -1006.5
Df Residuals:       31835                          LLR p-value:      nan
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     9.0000
--------------------------------------------------------------------------------
          Coef.        Std.Err.          z          P>|z|         [0.025        0.975]
--------------------------------------------------------------------------------
F3       -0.0065         0.0001       -73.7765       0.0000       -0.0066       -0.0063
==========================================================================
```

```
logit_model=sm.Logit(y,X[['F2']])
result=logit_model.fit()
print(result.summary2())
```

```
================================================================================
Model:              Logit                          Pseudo R-squared: -18.466
Dependent Variable: FLAG di sana in liquid + anomala AIC:              39187.3786
Date:               2020-11-13 22:54               BIC:              39195.7470
No. Observations:   31836                          Log-Likelihood:   -19593.
Df Model:           0                              LL-Null:          -1006.5
Df Residuals:       31835                          LLR p-value:      nan
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     5.0000
--------------------------------------------------------------------------------
          Coef.        Std.Err.          z          P>|z|         [0.025        0.975]
--------------------------------------------------------------------------------
F2       -0.0013         0.0000       -60.3746       0.0000       -0.0014       -0.0013
================================================================================
```

F2 è il primo fattore che presenta segno corretto, dunque rimane nel modello; a questo punto, con l'inserimento del prossimo fattore (F6) si deve verificare che entrambi abbiano segno corretto:

```
logit_model=sm.Logit(y,X[['F2','F6']])
result=logit_model.fit()
print(result.summary2())
```

```
================================================================================
Model:              Logit                          Pseudo R-squared: -2.417
Dependent Variable: FLAG di sana in liquid + anomala AIC:              6882.0148
Date:               2020-11-13 22:54               BIC:              6898.7515
No. Observations:   31836                          Log-Likelihood:   -3439.0
Df Model:           1                              LL-Null:          -1006.5
Df Residuals:       31834                          LLR p-value:      1.0000
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     9.0000
--------------------------------------------------------------------------------
             Coef.       Std.Err.          z          P>|z|         [0.025        0.975]
--------------------------------------------------------------------------------
F2          -0.0100        0.0002      -61.7770       0.0000       -0.0103       -0.0097
F6           0.0185        0.0003       70.2844       0.0000        0.0180        0.0190
================================================================================
```

Entrambi hanno segno corretto; si procede con i prossimi inserimenti:

```
logit_model=sm.Logit(y,X[['F2','F6','F7']])
result=logit_model.fit()
print(result.summary2())
```

```
================================================================================
Model:              Logit                          Pseudo R-squared: -2.413
Dependent Variable: FLAG di sana in liquid + anomala AIC:              6875.4090
Date:               2020-11-13 22:54               BIC:              6900.5141
No. Observations:   31836                          Log-Likelihood:   -3434.7
Df Model:           2                              LL-Null:          -1006.5
Df Residuals:       31833                          LLR p-value:      1.0000
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     9.0000
--------------------------------------------------------------------------------
            Coef.       Std.Err.         z          P>|z|        [0.025       0.975]
--------------------------------------------------------------------------------
F2          -0.0100        0.0002      -61.3658      0.0000      -0.0104      -0.0097
F6           0.0186        0.0003       69.8803      0.0000       0.0180       0.0191
F7           0.0009        0.0003        2.8850      0.0039       0.0003       0.0015
================================================================================
```

```
logit_model=sm.Logit(y,X[['F2','F6','F7','F8']])
result=logit_model.fit()
print(result.summary2())
```

```
================================================================================
Model:              Logit                          Pseudo R-squared: -0.721
Dependent Variable: FLAG di sana in liquid + anomala AIC:              3471.6694
Date:               2020-11-13 22:54               BIC:              3505.1428
No. Observations:   31836                          Log-Likelihood:   -1731.8
Df Model:           3                              LL-Null:          -1006.5
Df Residuals:       31832                          LLR p-value:      1.0000
Converged:          1.0000                         Scale:            1.0000
No. Iterations:     10.0000
--------------------------------------------------------------------------------
            Coef.       Std.Err.         z          P>|z|        [0.025       0.975]
--------------------------------------------------------------------------------
F2          -0.0099        0.0002      -41.6671      0.0000      -0.0103      -0.0094
F6           0.0174        0.0003       52.1298      0.0000       0.0167       0.0181
F7           0.0015        0.0004        3.3713      0.0007       0.0006       0.0024
F8          -0.0110        0.0004      -25.9779      0.0000      -0.0118      -0.0101
================================================================================
```

In questa 2° versione, il modello mantiene F2 ma si introducono anche i fattori F6, F7 e F8:

```
X= X[['F2','F6','F7','F8']]
X
```

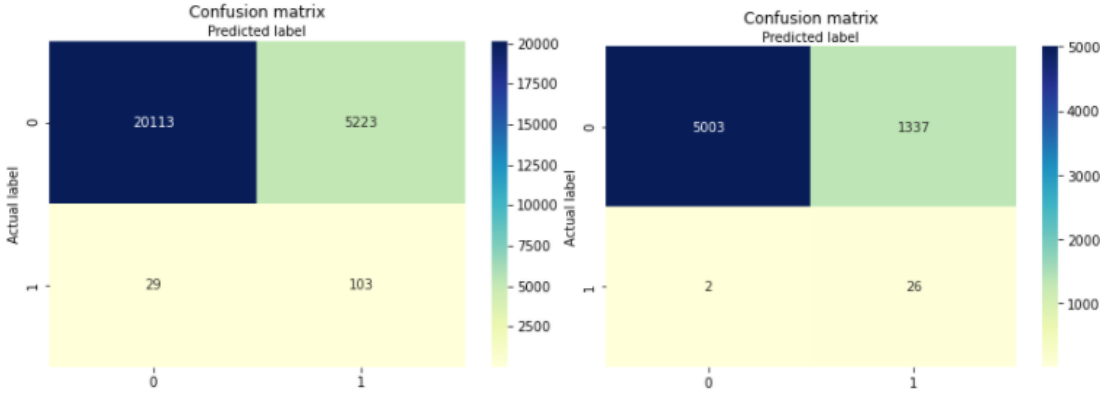
	F2	F6	F7	F8
0	-89.851876	-439.434362	33.152268	18.181339
1	182.194206	-200.806626	17.090908	-36.775821
2	103.475428	-258.221130	19.842617	-31.154059
3	136.844391	-276.143021	41.186747	-19.933704
4	146.269148	-237.419107	17.099149	-17.470253
...	...	...	...	...
31831	-1810.479229	-1602.446328	5.046828	-202.533574
31832	-930.368347	-941.701823	592.270261	-38.695339
31833	1688.548290	-519.404734	605.686884	455.372048
31834	1650.576934	-501.594085	638.181848	449.746742
31835	1659.375184	-515.761004	601.886176	446.323393

31836 rows × 4 columns

Risultati tuning e convalida:

```
Best: 0.765152 using {'solver': 'lbfgs', 'penalty': 'none'}

Fold 1 score=0.7990
Fold 2 score=0.8033
Fold 3 score=0.7868
Fold 4 score=0.7888
Fold 5 score=0.7856
Fold 6 score=0.8131
Fold 7 score=0.8025
Fold 8 score=0.7951
Fold 9 score=0.7781
Fold 10 score=0.7981

Validation Accuracy = 0.80
```

Si procede con l'addestramento sull'intero training set; di seguito sono riportati i risultati:



```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.79	0.88	25336
1	0.02	0.78	0.04	132
accuracy			0.79	25468
macro avg	0.51	0.79	0.46	25468
weighted avg	0.99	0.79	0.88	25468

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.79	0.88	6340
1	0.02	0.93	0.04	28
accuracy			0.79	6368
macro avg	0.51	0.86	0.46	6368
weighted avg	1.00	0.79	0.88	6368

Dal confronto dei risultati delle due versioni si evince che l'immissione nel modello di altri 3 fattori -in aggiunta a F2 comune a entrambi- ha portato a un significativo aumento della recall sulla classe 1: sul train si riesce a catturare il 78% dei casi anomali (contro il 73% della prima versione) e sul test set, addirittura, si arriva a classificarne correttamente 26 su 28 totali, registrando un richiamo di circa il 93%. Tuttavia, questo fatto palesa un chiaro problema in entrambe le versioni: l'underfitting; infatti come riscontrato, la recall sulla classe positiva registra un forte aumento sia nella prima che nella seconda versione rispettivamente di 13 e 15 punti percentuali; decisamente eccessivi rispetto ai risultati ottenuti nel training set. Le performance sugli altri indicatori sono pressoché le medesime.

In seguito, si ripropongono le due versioni utilizzando la target *società*.

[1° versione] (target *società*)

Il processo rimane lo stesso ma cambiando la target si vanno a modificare i coefficienti di correlazione:

```
#COEFF DI CORRELAZIONE TRA OGNI FATTORE E LA y:
for i in range(len(cols)):
    print(np.corrcoef(X[cols[i]],y['FLAG di sana in liquid + anomala (società)'])[0][1], cols[i])
```

```
-0.45258061682809114 F1
-0.2933044514498313 F2
0.46440868673535585 F3
-0.41050903580338044 F4
0.4232901753000158 F5
-0.38255691417391774 F6
0.021873489183030892 F7
0.031418345284296316 F8
```

Anche in questo caso i fattori da eliminare risultano F6 e F8; il processo iterativo con il quale si rimuovono man mano i fattori non idonei, in questa prima versione abbinata alla target *società*, porta ad ottenere un modello composto da:

```
==============================================================================================
Model:              Logit                                  Pseudo R-squared: -0.331
Dependent Variable: FLAG di sana in liquid + anomala (società) AIC:              36194.7922
Date:               2020-11-13 17:53                       BIC:              36211.5289
No. Observations:   31836                                  Log-Likelihood:   -18095.
Df Model:           1                                      LL-Null:          -13598.
Df Residuals:       31834                                  LLR p-value:      1.0000
Converged:          1.0000                                 Scale:            1.0000
No. Iterations:     5.0000
----------------------------------------------------------------------------------------------
            Coef.          Std.Err.            z             P>|z|           [0.025          0.975]
----------------------------------------------------------------------------------------------
F2         -0.0017           0.0000         -64.5254         0.0000         -0.0018         -0.0017
F4         -0.0005           0.0000         -17.2455         0.0000         -0.0006         -0.0005
==============================================================================================
```

```
X
```

	F2	F4
0	-89.851876	-540.040705
1	182.194206	78.522557
2	103.475428	53.502587
3	136.844391	196.323594
4	146.269148	198.294481
...	...	...
31831	-1810.479229	-413.939291
31832	-930.368347	114.735997
31833	1688.548290	12.816183
31834	1650.576934	-127.028692
31835	1659.375184	13.004462

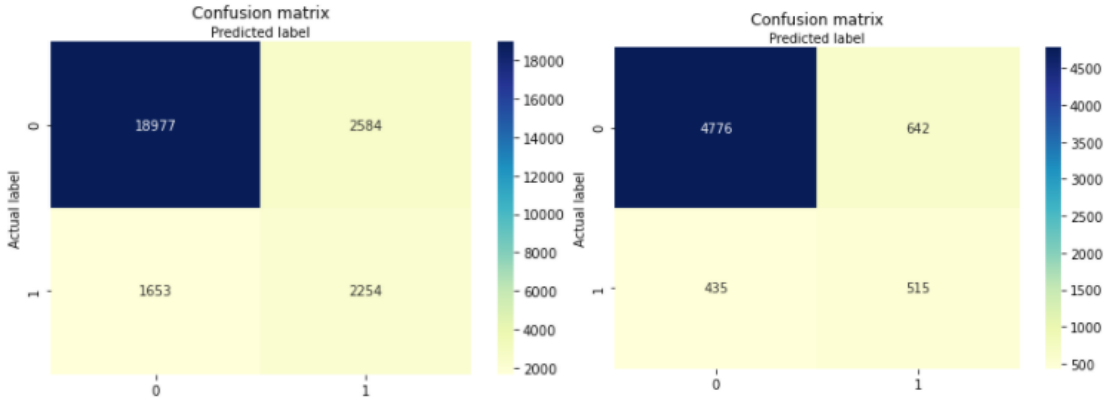
31836 rows × 2 columns

La fasi di ottimizzazione e convalida si concludono con i seguenti risultati:

```
Best: 0.576138 using {'solver': 'lbfgs', 'penalty': 'none'}
```

```
Fold 1 score=0.8382
Fold 2 score=0.8355
Fold 3 score=0.8422
Fold 4 score=0.8386
Fold 5 score=0.8253
Fold 6 score=0.8335
Fold 7 score=0.8320
Fold 8 score=0.8292
Fold 9 score=0.8189
Fold 10 score=0.8409

Validation Accuracy = 0.83
```

I risultati su train e test sono:




```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	0.92	0.88	0.90	21561
1	0.47	0.58	0.52	3907
accuracy			0.83	25468
macro avg	0.69	0.73	0.71	25468
weighted avg	0.85	0.83	0.84	25468

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.92	0.88	0.90	5418
1	0.45	0.54	0.49	950
accuracy			0.83	6368
macro avg	0.68	0.71	0.69	6368
weighted avg	0.85	0.83	0.84	6368

[2° versione] (target *società*)

Al contrario del precedente, il modello nella sua 2° versione implementa un numero di fattori pari a 5:

```
==============================================================================================
Model:              Logit                                  Pseudo R-squared: -0.079
Dependent Variable: FLAG di sana in liquid + anomala (società) AIC:              29347.2061
Date:               2020-11-11 18:32                       BIC:              29389.0479
No. Observations:   31836                                  Log-Likelihood:   -14669.
Df Model:           4                                      LL-Null:          -13598.
Df Residuals:       31831                                  LLR p-value:      1.0000
Converged:          1.0000                                 Scale:            1.0000
No. Iterations:     6.0000
----------------------------------------------------------------------------------------------------
            Coef.          Std.Err.            z             P>|z|          [0.025          0.975]
----------------------------------------------------------------------------------------------------
F2         -0.0022           0.0000         -53.8264         0.0000         -0.0023         -0.0021
F4         -0.0013           0.0000         -38.1363         0.0000         -0.0014         -0.0012
F6          0.0031           0.0000          63.4875         0.0000          0.0030          0.0032
F7          0.0009           0.0001           7.7041         0.0000          0.0007          0.0011
F8         -0.0014           0.0001         -19.5949         0.0000         -0.0016         -0.0013
==============================================================================================
```

```
X= X[['F2','F4','F6','F7','F8']]
X
```

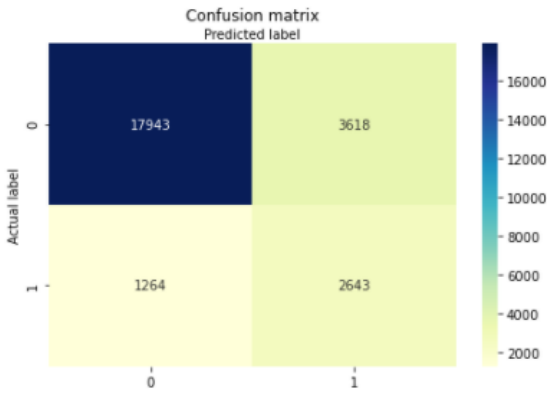
	F2	F4	F6	F7	F8
0	-89.851876	-540.040705	-439.434362	33.152268	18.181339
1	182.194206	78.522557	-200.806626	17.090908	-36.775821
2	103.475428	53.502587	-258.221130	19.842617	-31.154059
3	136.844391	196.323594	-276.143021	41.186747	-19.933704
4	146.269148	198.294481	-237.419107	17.099149	-17.470253
...	...	...	...	...	...
31831	-1810.479229	-413.939291	-1602.446328	5.046828	-202.533574
31832	-930.368347	114.735997	-941.701823	592.270261	-38.695339
31833	1688.548290	12.816183	-519.404734	605.686884	455.372048
31834	1650.576934	-127.028692	-501.594085	638.181848	449.746742
31835	1659.375184	13.004462	-515.761004	601.886176	446.323393

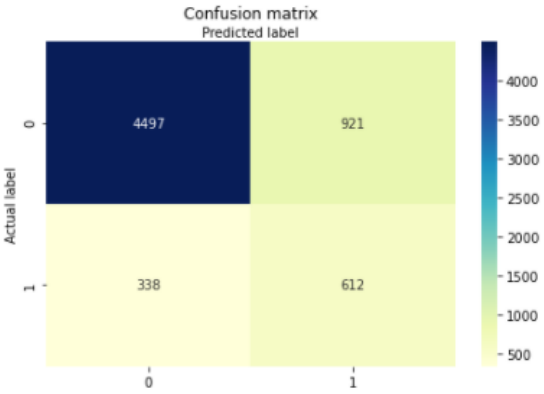
31836 rows × 5 columns

Ottimizzazione e convalida:

```
Best: 0.677754 using {'solver': 'lbfgs', 'penalty': 'none'}


Fold 1 score=0.8155
Fold 2 score=0.8131
Fold 3 score=0.8135
Fold 4 score=0.8057
Fold 5 score=0.8013
Fold 6 score=0.8053
Fold 7 score=0.8053
Fold 8 score=0.8072
Fold 9 score=0.7969
Fold 10 score=0.8150

Validation Accuracy = 0.81
```

Di seguito i risultati finali:






```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	0.93	0.83	0.88	21561
1	0.42	0.68	0.52	3907
accuracy			0.81	25468
macro avg	0.68	0.75	0.70	25468
weighted avg	0.86	0.81	0.82	25468

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.93	0.83	0.88	5418
1	0.40	0.64	0.49	950
accuracy			0.80	6368
macro avg	0.66	0.74	0.69	6368
weighted avg	0.85	0.80	0.82	6368

Gli ultimi 4 modelli di analisi fattoriale si possono analizzare sotto il punto di vista della versione e della target scelta; per quanto concerne quest'ultima, come riscontrato nel caso della *logistica base*, i modelli che utilizzano la target di tipo *società* risultano essere complessivamente molto più equilibrati in termini di score f1, in quanto riescono a raggiungere livelli di precisione sulla classe delle anomale decisamente più alti dei modelli che hanno per oggetto la risposta *anno*. Nonostante ciò, l'elevato numero di falsi negativi ancora presenti -a cause delle scarse performance del richiamo sulla classe positiva- compromette inevitabilmente la loro utilità complessiva; dunque, data l'importanza di mantenere al minimo l'errore di secondo tipo, anche con l'implementazione dell'analisi fattoriale la scelta della target *anno* risulta la migliore per il credit scoring.

Restando sulla risposta *anno*, le due versioni hanno performance simili ma la seconda presenta una recall sulla classe 1 sistematicamente superiore alla prima; infatti, sia sul train che sul test set quest'ultima, pur registrando

complessivamente discreti risultati, raggiunge rispettivamente un massimo del 73% e 86%, contro il 78% e 93% della seconda metodologia, corrispondenti a 7 e 2 casi in meno di falsi negativi sui 132 (train) e 28 (test) totali. Infine, mentre la prima versione produce un modello composto solo dal fattore F2, la seconda, permettendo l'introduzione di altre variabili, rende il modello complessivamente più attendibile e, quindi, preferibile.

Dunque, se si effettua una valutazione al netto del problema di underfitting descritto in precedenza, utilizzando l'analisi fattoriale il modello di regressione logistica migliore risulta essere quello costruito con la target *anno* nella sua 2° versione.

Rete Neurale Artificiale

In questa nuova sezione si abbandona la regressione logistica per affrontare una nuova metodologia che negli ultimi anni, con il rapido sviluppo del Deep Learning, è stata sempre più utilizzata in merito all'analisi di credit scoring: la *rete neurale artificiale*.

Anche in questo caso vengono presentati due modelli, il primo con la target *anno* e il secondo con la target *società*.

(target *anno*)

Il punto di partenza è ancora il dataset da 31836 osservazioni, 46 features e 1 target, memorizzato nel dataframe pandas *credit*:

	FLAG di sana in liquid + anomala	EBITDA/Ricavi	EBIT/ricavi	Utile corrente/ricavi	...	Ebitda-servizio debito/AN	Ln(AN)	Ln(RIC)
0	0	-3.44	-13.47	-8.65		-5.34	11.10	11.11
1	0	2.72	-5.56	1.43		-0.71	11.10	11.11
2	0	0.64	-5.86	-6.01		-2.27	11.10	11.11
3	0	6.16	3.15	1.54		-0.16	11.10	11.11
4	0	5.88	3.03	2.48		2.58	11.10	11.11
...	...	...	...	...		...	...	...
31831	0	33.26	24.37	0.00		-18.58	4.39	1.10
31832	0	33.26	24.37	0.00		-25.23	4.39	1.10
31833	0	33.26	24.37	0.00		-25.23	4.39	1.10
31834	0	33.26	0.00	-33.33		-25.23	4.39	1.10
31835	0	33.26	24.37	0.00		-4.88	4.77	1.10

31836 rows × 47 columns

Con lo stesso processo di features selection implementato nel primo modello "Regressione Logistica Base (target *anno*)", si rimuovono le features:

```
elim= ['EBITDA/Ricavi', 'EBIT/ricavi', 'Risultato netto rettif/ricavi',
       'EBITDA/AN', 'ROE ante imposte', 'AC-mag/PC', 'Patr netto/AN',
       'Patr netto tan/Debiti tot+PN', 'Patr netto tan/Debiti tot-Liq+PN',
       'OF/RIC', 'OFN/EBITDA', 'Autof Lordo/AN (=cash flow/attivo)',
       'Deb totali/Ric', 'Debiti totali/EBITDA']

credit= credit.drop(elim, axis=1)
```

ottenendo il seguente dataset e target:

```
X = credit.loc[:, credit.columns != 'FLAG di sana in liquid + anomala']
y = credit.loc[:, credit.columns == 'FLAG di sana in liquid + anomala']

print('Shape of X: {}'.format(X.shape))
print('Shape of y: {}'.format(y.shape))
```

```
Shape of X: (31836, 32)
Shape of y: (31836, 1)
```

```
X
```

	Utile corrente/ricavi	Risultato netto rettif/AN	ROA	ROE	...	Ebitda-servizio debito/AN	Ln(AN)	Ln(RIC)
0	-8.65	-6.82	-2.65	-19.40		-5.34	11.10	11.11
1	1.43	0.64	2.08	1.82		-0.71	11.10	11.11
2	-6.01	-0.48	-3.32	-1.50		-2.27	11.10	11.11
3	1.54	-0.30	3.57	-1.00		-0.16	11.10	11.11
4	2.48	0.99	4.80	3.47		2.58	11.10	11.11
...	...	...	...	...		...	...	...
31831	0.00	0.00	3.23	-200.00		-18.58	4.39	1.10
31832	0.00	0.00	12.50	-200.00		-25.23	4.39	1.10
31833	0.00	0.00	11.11	0.00		-25.23	4.39	1.10
31834	-33.33	-11.11	0.00	-12.50		-25.23	4.39	1.10
31835	0.00	0.00	0.85	0.00		-4.88	4.77	1.10

31836 rows × 32 columns

```
y  #y_anno
```

	FLAG di sana in liquid + anomala
0	0
1	0
2	0
3	0
4	0
...	...
31831	0
31832	0
31833	0
31834	0
31835	0

31836 rows × 1 columns

Si completa la fase di preprocessing con lo split e normalizzazione del dataset:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123456789)

ss= MinMaxScaler()
X_train= ss.fit_transform(X_train)
X_test= ss.transform(X_test)

# describes info about train and test set
print("X_train dataset: ", X_train.shape)
print("y_train dataset: ", y_train.shape)
print("X_test dataset: ", X_test.shape)
print("y_test dataset: ", y_test.shape)
```

```
X_train dataset:  (25468, 32)
y_train dataset:  (25468, 1)
X_test dataset:  (6368, 32)
y_test dataset:  (6368, 1)
```

A questo punto si procede con la fase di tuning degli iperparametri. È importante sottolineare come non esista un unico modello standard valido per tutte i tipi di problemi e, al contempo, come uno stesso problema possa essere affrontato con diversi modelli; l'obiettivo, dunque, è quello di trovare un insieme di parametri che permettano di estrarre le migliori performance dalla rete. Inizialmente, viene creata una funzione, chiamata *create_model()*, che restituisce una rete neurale artificiale *compilata*:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.regularizers import l2
from keras.layers import Dropout
from sklearn import metrics

def create_model(neurons1=1, neurons2=1, neurons3=1, neurons4=1, dropout_rate=0.5, optimizer='adam'):
    # create model
    model = Sequential()
    model.add(Dense(neurons1,activation='relu',input_shape=(X_train.shape[1],),kernel_regularizer=l2(0.1)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(neurons2,activation='relu',kernel_regularizer=l2(0.01)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(neurons3,activation='relu',kernel_regularizer=l2(0.001)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(neurons4,activation='relu', kernel_regularizer=l2(0.01)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model
```

Attraverso la classe *sequential* si istanzia un nuovo modello sequenziale di rete neurale. I seguenti iperparametri sono stati settati *manualmente* al fine di agevolare l'ottimizzazione e di ridurre la mole di calcolo richiesta al processore:

- *Numero di livelli della rete*: si tratta di un modello composto da 4 strati nascosti; inseriti tramite il metodo *add()*, questi sono tutti della tipologia "denso", ovvero ogni nodo dello strato di partenza è collegato a ogni nodo dello strato immediatamente successivo. Attraverso il parametro *input_shape*[23] viene inserito il numero di nodi dello strato di input; questo è inserito esclusivamente nel primo strato in quanto nei successivi sarà la rete stessa a rilevarlo automaticamente dallo strato precedente;
- *Funzione di attivazione*: come dimostrato in letteratura, viene utilizzata la 'relu' per i vari strati nascosti e la 'sigmoide' nell'ultimo strato (che fornirà l'output), composto da un unico neurone, in quanto il problema è di classificazione binaria;
- *Regolarizzazione*: si è optato per una penalizzazione di tipo 'l2', invece che la l1, in quanto quest'ultima avrebbe introdotto una seconda e superflua features selection. La regolarizzazione viene applicata ai pesi di ogni strato della rete attraverso l'argomento *kernel_regularizer*, al quale si passa come valore la funzione importata *l2* specificando al suo interno l'entità dell'iperparametro λ; come si vede, si è impostata un'intensità maggiore nel primo strato rispetto agli altri.

La configurazione della fase di addestramento viene effettuata utilizzando il metodo *compile()* nel quale si specifica la *binary_crossentropy* come funzione di costo da minimizzare e l'*accuracy* come metrica da calcolare.

Si procede con la definizione della fase di tuning degli iperparametri:

[23] Nel caso in esame X_train.shape[1] mostra il numero di colonne/features del dataframe X, quindi 32.

```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import RandomizedSearchCV

# create model
model = KerasClassifier(build_fn=create_model, verbose=0)

# define the grid search parameters
space = dict()
space['neurons1'] = [64 ,128, 256]
space['neurons2'] = [32, 64, 128]
space['neurons3'] = [16, 32, 64]
space['neurons4'] = [8, 16, 32]
space['batch_size'] = [64, 128, 256]
space['dropout_rate'] = [0.1, 0.2, 0.3, 0.4, 0.5]
space['optimizer'] = ['Adam', 'Adamax', 'Nadam']

class_weight = {0: 0.5026, 1: 96.47}

grid = RandomizedSearchCV(model, space, scoring='recall', n_jobs=-1, cv=3)
random_result = grid.fit(X_train, y_train, epochs=100, class_weight=class_weight)
```

La libreria *keras* fornisce un comodo "wrapper" (avvolgimento) ai modelli di Deep Learning in modo tale che questi possano essere utilizzati da scikit-learn; questa attività di wrapping sul modello permette di sfruttare i potenti strumenti e funzionalità di scikit-learn, compresa la *Random Search* e la *k-fold cross validation* (eseguita in seguito), che verranno applicate al modello stesso. Nella classe *KerasClassifier* viene settato l'argomento *build_fn* il quale contiene il nome della funzione da richiamare per ottenere il modello. Gli iperparametri che si è scelto di ottimizzare sono:

- *Numero di neuroni per ogni hidden layer*: si è seguita la convenzione di impostare uno spazio di ricerca contenente valori multipli di 2;
- *batch_size*: questo è uno degli iperparametri che maggiormente influenza i risultati dell'apprendimento; lo space arriva fino a 256 per evitare problemi di scarsità di memoria[24] del processore. Generalmente, i professionisti preferiscono un batch_size di grandi dimensioni (512,

[24] Una dimensione eccessivamente grande può portare a una scarsa generalizzazione, a problemi di minimo locali e di memoria. Al contrario, l'uso di lotti di dimensioni inferiori consente al modello di "iniziare ad apprendere prima di dover vedere tutti i dati", guidandolo a una convergenza più rapida verso buone soluzioni; lo svantaggio, se le dimensioni sono troppo piccole, è rappresentato dal fatto che non è garantita la converga all'ottimo globale.

1024, 2048, …) in quanto consente una fase di addestramento sensibilmente più veloce;

- *Dropout*: al fine di ridurre la probabilità che il modello soffra di *overfitting*[25], si introduce, oltre alla penalità l2, questa ulteriore regolarizzazione; i valori dello space rappresentano la percentuale di nodi da rimuovere a ogni iterazione (impostato fino a un massimo di 0.5, ovvero la metà dei nodi per ogni livello); tale comando deve essere inserito dopo lo strato sul quale si vuole applicare la regolarizzazione.
- *Ottimizzatore*: si è optato per uno space contenente esclusivamente l'Adam e le sue varianti (Adamax e Nadam) in quanto risultano dalla letteratura i migliori algoritmi di ottimizzazione.

Come visto durante i modelli di regressione logistica, si istanzia il metodo RandomizedSearchCV e se ne esegue l'addestramento impostando *manualmente* i pesi delle classi (in modo tale da renderle bilanciate[26]) e il numero di epoche:

```
Best: 0.856061 using {'optimizer': 'Nadam', 'neurons4': 8, 'neurons3': 16, 'neurons2': 32, 'neurons
1': 128, 'dropout_rate': 0.4, 'batch_size': 256}
```

A questo punto si effettua la validazione su una rete neurale composta da 4 strati nascosti di rispettivamente 128, 32, 16, 8 neuroni, un dropout_rate di 0.4 e addestrata, con un batch_size di 256, utilizzando l'algoritmo Nadam.

[25] Overfitting: quando la rete, legandosi eccessivamente ai dati di addestramento, non riesce a generalizzare su nuovi dati le regole apprese durante il training.

[26] I pesi delle classi per un modello con target *anno* sono stati calcolati a pp. 144

```
def rna():
    model = Sequential()
    model.add(Dense(128,activation='relu',input_shape=(X_train.shape[1],),kernel_regularizer=l2(0.1)))
    model.add(Dropout(0.4))
    model.add(Dense(32,activation='relu', kernel_regularizer=l2(0.01)))
    model.add(Dropout(0.4))
    model.add(Dense(16,activation='relu',kernel_regularizer=l2(0.001)))
    model.add(Dropout(0.4))
    model.add(Dense(8,activation='relu', kernel_regularizer=l2(0.01)))
    model.add(Dropout(0.4))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer='nadam', loss='binary_crossentropy', metrics=['accuracy'])

    return model

rna_model= KerasClassifier(build_fn=rna, epochs=100, batch_size=256)
```

Il modello viene "avvolto" e passato in input alla classe cross_val_score di scikit-learn:

```
from sklearn.model_selection import cross_val_score

scores= cross_val_score(rna_model, X_train, y_train, cv=10,
                        fit_params = {'class_weight': {0: 0.5026, 1: 96.47}})

for fold,score in enumerate(scores):
    print("Fold %d score=%.4f" % (fold+1,score))

print("\nValidation Accuracy = %.2f" % scores.mean())
```

Al termine del lungo processo di addestramento delle 10 folds si ottengono in output le rispettive 10 accuracy:

```
Fold 1 score=0.7126
Fold 2 score=0.7456
Fold 3 score=0.8551
Fold 4 score=0.8398
Fold 5 score=0.7601
Fold 6 score=0.8280
Fold 7 score=0.8245
Fold 8 score=0.8740
Fold 9 score=0.7769
Fold 10 score=0.7765

Validation Accuracy = 0.80
```

Il modello di rete neurale multistrato appena descritto risulta convalidato. Prima di eseguire l'addestramento sull'intero training set e visualizzare i risultati, si mostra un riepilogo della RNA ottimizzata e convalidata:

```
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_51 (Dense)             (None, 128)               4224
_________________________________________________________________
dropout_41 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_52 (Dense)             (None, 32)                4128
_________________________________________________________________
dropout_42 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_53 (Dense)             (None, 16)                528
_________________________________________________________________
dropout_43 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_54 (Dense)             (None, 8)                 136
_________________________________________________________________
dropout_44 (Dropout)         (None, 8)                 0
_________________________________________________________________
dense_55 (Dense)             (None, 1)                 9
=================================================================
Total params: 9,025
Trainable params: 9,025
Non-trainable params: 0
_________________________________________________________________
```

Con i suoi 4 strati nascosti e 1 di output, la rete risulta avere 9025 *parametri da addestrare*, ovvero:

$$128 + (128 \cdot 32) + 32 + (32 \cdot 128) + 16 + (16 \cdot 32) + 8 + (8 \cdot 16) + 1 + (1 \cdot 8) = 9025$$

la somma tra il numero di bias di ogni strato (corrispondente al numero di nodi dello strato) e il prodotto tra il numero di nodi di uno strato nascosto e del suo precedente. Ad esempio, la prima somma rappresenta i pesi che collegano i 32 nodi dello strato di input a ognuno dei 128 nodi del primo strato nascosto (128·32) e un bias per ogni nodo del primo strato nascosto (128).

A questo punto, si procede con l'addestramento composto da 100 epoche, con un batch_size pari a 256:

```
from keras.callbacks import History
from time import time

history= History()
class_weight = {0: 0.5026, 1: 96.47}

start_at= time()

model.fit(X_train, y_train, epochs=100, batch_size=256,
          class_weight=class_weight,callbacks=[history])

exec_time= time()-start_at
```

```
Epoch 1/100
25468/25468 [==============================] - 1s 43us/step - loss: 2.8418 - accuracy: 0.5786
Epoch 2/100
25468/25468 [==============================] - 1s 21us/step - loss: 0.8755 - accuracy: 0.6515
Epoch 3/100
25468/25468 [==============================] - 1s 22us/step - loss: 0.7255 - accuracy: 0.7586
Epoch 4/100
25468/25468 [==============================] - 1s 22us/step - loss: 0.6726 - accuracy: 0.8076
Epoch 5/100
25468/25468 [==============================] - 1s 23us/step - loss: 0.6233 - accuracy: 0.8100
Epoch 6/100
25468/25468 [==============================] - 1s 22us/step - loss: 0.6221 - accuracy: 0.8151
Epoch 7/100
25468/25468 [==============================] - 1s 22us/step - loss: 0.6047 - accuracy: 0.8136
Epoch 8/100
25468/25468 [==============================] - 1s 24us/step - loss: 0.6006 - accuracy: 0.7991
Epoch 9/100
25468/25468 [==============================] - 1s 22us/step - loss: 0.6210 - accuracy: 0.7774
Epoch 10/100
25468/25468 [==============================] - 1s 22us/step - loss: 0.5823 - accuracy: 0.8019
Epoch 11/100
25468/25468 [==============================] - 1s 22us/step - loss: 0.6042 - accuracy: 0.7671
Epoch 12/100
25468/25468 [==============================] - 1s 24us/step - loss: 0.5927 - accuracy: 0.7779
Epoch 13/100
25468/25468 [==============================] - 1s 28us/step - loss: 0.5599 - accuracy: 0.8004
Epoch 14/100
25468/25468 [==============================] - 1s 33us/step - loss: 0.5937 - accuracy: 0.7942
Epoch 15/100
25468/25468 [==============================] - 1s 29us/step - loss: 0.5585 - accuracy: 0.7983
Epoch 16/100
25468/25468 [==============================] - 1s 43us/step - loss: 0.5514 - accuracy: 0.8175
Epoch 17/100
25468/25468 [==============================] - 1s 28us/step - loss: 0.5662 - accuracy: 0.7972
Epoch 18/100
25468/25468 [==============================] - 1s 28us/step - loss: 0.5825 - accuracy: 0.8050
Epoch 19/100
25468/25468 [==============================] - 1s 29us/step - loss: 0.5421 - accuracy: 0.8194
Epoch 20/100
25468/25468 [==============================] - 1s 28us/step - loss: 0.5916 - accuracy: 0.7627
```

```
Epoch 21/100
25468/25468 [==============================] - 1s 38us/step - loss: 0.5835 - accuracy: 0.7923
Epoch 22/100
25468/25468 [==============================] - 1s 34us/step - loss: 0.5604 - accuracy: 0.8101
Epoch 23/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5748 - accuracy: 0.7983
Epoch 24/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5545 - accuracy: 0.7898
Epoch 25/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5303 - accuracy: 0.7999
Epoch 26/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5555 - accuracy: 0.8275
Epoch 27/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5507 - accuracy: 0.8110
Epoch 28/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5381 - accuracy: 0.8045
Epoch 29/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5147 - accuracy: 0.8164
Epoch 30/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5606 - accuracy: 0.7994
Epoch 31/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5522 - accuracy: 0.7764
Epoch 32/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5533 - accuracy: 0.8113
Epoch 33/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5387 - accuracy: 0.7945
Epoch 34/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5256 - accuracy: 0.8034
Epoch 35/100
25468/25468 [==============================] - 1s 29us/step - loss: 0.5524 - accuracy: 0.7970
Epoch 36/100
25468/25468 [==============================] - 1s 28us/step - loss: 0.5890 - accuracy: 0.7723
Epoch 37/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5449 - accuracy: 0.7963
Epoch 38/100
25468/25468 [==============================] - 1s 28us/step - loss: 0.5506 - accuracy: 0.8016
Epoch 39/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5593 - accuracy: 0.7926
Epoch 40/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5444 - accuracy: 0.7849
```

```
Epoch 41/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5639 - accuracy: 0.7939
Epoch 42/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5463 - accuracy: 0.8072
Epoch 43/100
25468/25468 [==============================] - 1s 27us/step - loss: 0.5291 - accuracy: 0.7999
Epoch 44/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5343 - accuracy: 0.8053
Epoch 45/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5639 - accuracy: 0.7782
Epoch 46/100
25468/25468 [==============================] - 1s 29us/step - loss: 0.5554 - accuracy: 0.8001
Epoch 47/100
25468/25468 [==============================] - 1s 29us/step - loss: 0.5452 - accuracy: 0.8011
Epoch 48/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5383 - accuracy: 0.7921
Epoch 49/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5422 - accuracy: 0.8008
Epoch 50/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5700 - accuracy: 0.8044
Epoch 51/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5571 - accuracy: 0.7985
Epoch 52/100
25468/25468 [==============================] - 1s 27us/step - loss: 0.5550 - accuracy: 0.7943
Epoch 53/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5140 - accuracy: 0.8125
Epoch 54/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5461 - accuracy: 0.8187
Epoch 55/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5378 - accuracy: 0.8113
Epoch 56/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5708 - accuracy: 0.7700
Epoch 57/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5497 - accuracy: 0.7841
Epoch 58/100
25468/25468 [==============================] - 1s 27us/step - loss: 0.5443 - accuracy: 0.7989
Epoch 59/100
25468/25468 [==============================] - 1s 29us/step - loss: 0.5423 - accuracy: 0.7650
Epoch 60/100
25468/25468 [==============================] - 1s 29us/step - loss: 0.5244 - accuracy: 0.8104
```

```
Epoch 61/100
25468/25468 [==============================] - 1s 28us/step - loss: 0.5157 - accuracy: 0.8201
Epoch 62/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5613 - accuracy: 0.7803
Epoch 63/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5497 - accuracy: 0.8166
Epoch 64/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5310 - accuracy: 0.8138
Epoch 65/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5357 - accuracy: 0.8077
Epoch 66/100
25468/25468 [==============================] - 1s 27us/step - loss: 0.5425 - accuracy: 0.8037
Epoch 67/100
25468/25468 [==============================] - 1s 27us/step - loss: 0.5289 - accuracy: 0.7964
Epoch 68/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5470 - accuracy: 0.8049
Epoch 69/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5025 - accuracy: 0.8215
Epoch 70/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5320 - accuracy: 0.8163
Epoch 71/100
25468/25468 [==============================] - 1s 27us/step - loss: 0.5502 - accuracy: 0.7907
Epoch 72/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5353 - accuracy: 0.8102
Epoch 73/100
25468/25468 [==============================] - 1s 27us/step - loss: 0.5169 - accuracy: 0.8055
Epoch 74/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5347 - accuracy: 0.7898
Epoch 75/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5225 - accuracy: 0.8026
Epoch 76/100
25468/25468 [==============================] - 1s 27us/step - loss: 0.5422 - accuracy: 0.7822
Epoch 77/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5422 - accuracy: 0.8069
Epoch 78/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5537 - accuracy: 0.8195
Epoch 79/100
25468/25468 [==============================] - 1s 24us/step - loss: 0.5458 - accuracy: 0.8062
Epoch 80/100
25468/25468 [==============================] - 1s 24us/step - loss: 0.5437 - accuracy: 0.8034
```

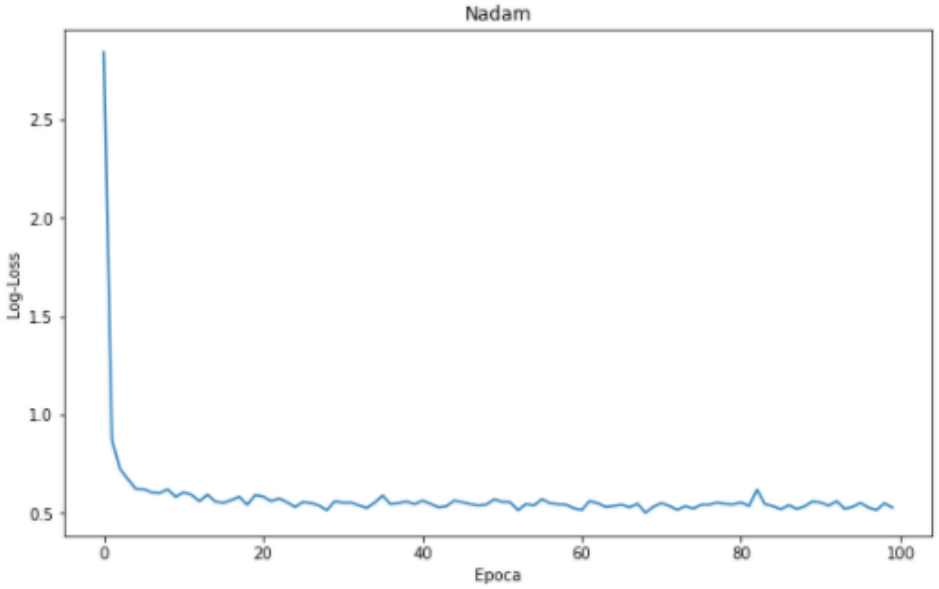
```
Epoch 81/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5543 - accuracy: 0.7682
Epoch 82/100
25468/25468 [==============================] - 1s 28us/step - loss: 0.5348 - accuracy: 0.7872
Epoch 83/100
25468/25468 [==============================] - 1s 29us/step - loss: 0.6186 - accuracy: 0.6613
Epoch 84/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5455 - accuracy: 0.7997
Epoch 85/100
25468/25468 [==============================] - 1s 30us/step - loss: 0.5345 - accuracy: 0.7906
Epoch 86/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5197 - accuracy: 0.8189
Epoch 87/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5394 - accuracy: 0.7929
Epoch 88/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5209 - accuracy: 0.8142
Epoch 89/100
25468/25468 [==============================] - 1s 25us/step - loss: 0.5341 - accuracy: 0.8059
Epoch 90/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5599 - accuracy: 0.7820
Epoch 91/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5530 - accuracy: 0.7830
Epoch 92/100
25468/25468 [==============================] - 1s 26us/step - loss: 0.5370 - accuracy: 0.8264
Epoch 93/100
25468/25468 [==============================] - 1s 27us/step - loss: 0.5606 - accuracy: 0.7855
Epoch 94/100
25468/25468 [==============================] - 1s 27us/step - loss: 0.5217 - accuracy: 0.8074
Epoch 95/100
25468/25468 [==============================] - 1s 27us/step - loss: 0.5311 - accuracy: 0.8140
Epoch 96/100
25468/25468 [==============================] - 1s 27us/step - loss: 0.5512 - accuracy: 0.7989
Epoch 97/100
25468/25468 [==============================] - 1s 27us/step - loss: 0.5276 - accuracy: 0.8310
Epoch 98/100
25468/25468 [==============================] - 1s 28us/step - loss: 0.5156 - accuracy: 0.8076
Epoch 99/100
25468/25468 [==============================] - 1s 28us/step - loss: 0.5494 - accuracy: 0.7975
Epoch 100/100
25468/25468 [==============================] - 1s 29us/step - loss: 0.5276 - accuracy: 0.7916
```

```
print("Tempo di addestramento: %d minuti e %d secondi" % (exec_time/60, exec_time%60))
```

```
Tempo di addestramento: 1 minuti e 10 secondi
```

```
plt.figure(figsize=(10,6))
plt.title('Nadam')
plt.xlabel('Epoca')
plt.ylabel('Log-Loss')
plt.plot(history.history['loss'])
```

```
[<matplotlib.lines.Line2D at 0x2643104f908>]
```



L'inserimento delle due tecniche di regolarizzazione (L2 e dropout) ha reso la curva della Log-Loss maggiormente lineare in modo tale da ottenere un modello stabile nelle sue predizioni.

Si valutano i risultati sul train e test set:

```
model.evaluate(X_train, y_train)
```

```
25468/25468 [==============================]
[0.5163314261709069, 0.8001021146774292]
```

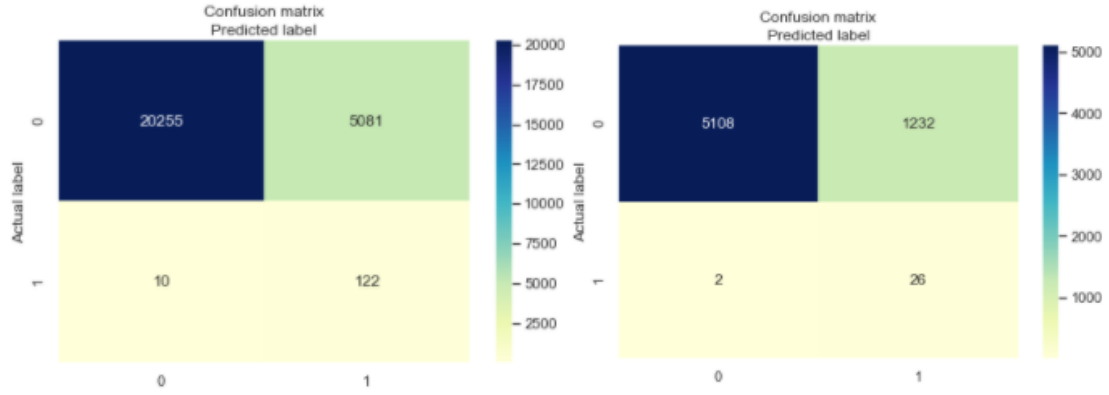
```
model.evaluate(X_test, y_test)
```

```
6368/6368 [==============================]
[0.5030720924312745, 0.8062185645103455]
```

Sul set di addestramento, la RNA raggiunge un valore della funzione di costo di 0.52 e un'accuracy di circa l'80%, impiegando 1 minuto e 10 secondi, mentre nel test set la valutazione è addirittura migliore in entrambe le metriche in quanto si riduce la Log-Loss e aumenta l'accuratezza. In generale, le due

evaluate con valori molto simili indicano che il modello non soffre di overfitting.

Di seguito si mostrano le performance sulla matrice di confusione del train e test set:




```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.80	0.89	25336
1	0.02	0.92	0.05	132
accuracy			0.80	25468
macro avg	0.51	0.86	0.47	25468
weighted avg	0.99	0.80	0.88	25468

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.81	0.89	6340
1	0.02	0.93	0.04	28
accuracy			0.81	6368
macro avg	0.51	0.87	0.47	6368
weighted avg	1.00	0.81	0.89	6368

L'insieme dei parametri selezionati dalla Random Search e passati per la 10-fold cross validation hanno creato una rete neurale molto performante, sia nel train che nel test, nella recall sulla classe delle anomale (ovvero l'indicatore che in questa analisi ricopre una maggiore importanza). Dunque, anche questo modello che utilizza la target *anno* risulta accettabile per un'analisi di credit scoring.

(target *società*)

Il modello con target *società* è rimasto invariato rispetto al precedente, con un'unica modifica riguardante i pesi, i quali vengono aggiustati per bilanciare le classi:

($n_{samples\_train}$ = 25468, $n_{classes}$ = 2, np.count(y_train) = [21561 ; 3907]) si ottiene:

{peso_classe_0 ; peso_classe_1} = {0.59 ; 3.26}

Si mostrano direttamente i risultati al termine della fase di tuning:

```
Best: 0.776473 using {'optimizer': 'Nadam', 'neurons4': 32, 'neurons3': 32, 'neurons2': 64, 'neurons
1': 256, 'dropout_rate': 0.4, 'batch_size': 256}
```

e di validazione incrociata:

```
Fold 1 score=0.8575
Fold 2 score=0.8100
Fold 3 score=0.8724
Fold 4 score=0.8170
Fold 5 score=0.8127
Fold 6 score=0.7436
Fold 7 score=0.8166
Fold 8 score=0.6835
Fold 9 score=0.8236
Fold 10 score=0.6622

Validation Accuracy = 0.79
```

Anche con l'utilizzo della target *società*, il modello rimane convalidato.

La fase di addestramento sul train set completo termina in:

```
print("Tempo di addestramento: %d minuti e %d secondi" % (exec_time/60, exec_time%60))
```

```
Tempo di addestramento: 1 minuti e 59 secondi
```

Di seguito la valutazione sul train e test set e le predizioni:

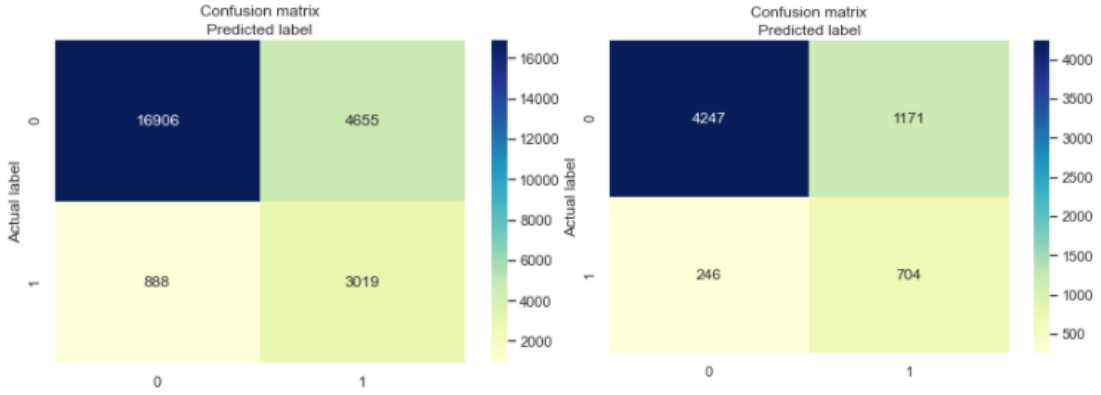
```
model.evaluate(X_train, y_train)
```

```
25468/25468 [==============================]
[0.5630651492047988, 0.7823543548583984]
```

```
model.evaluate(X_test, y_test)
```

```
6368/6368 [==============================]
[0.5687547809814089, 0.7774811387062073]
```

Da notare che entrambe le valutazioni sono inferiori a quelle ottenute nel modello *anno*.




```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	0.95	0.78	0.86	21561
1	0.39	0.77	0.52	3907
accuracy			0.78	25468
macro avg	0.67	0.78	0.69	25468
weighted avg	0.86	0.78	0.81	25468

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.95	0.78	0.86	5418
1	0.38	0.74	0.50	950
accuracy			0.78	6368
macro avg	0.66	0.76	0.68	6368
weighted avg	0.86	0.78	0.80	6368

Le performance ottenute con la target *società* sono perfettamente comparabili con quelle registrate utilizzando la regressione logistica (sia *base* sia *fattoriale*); infatti, il modello risulta avere una buon equilibrio tra le classi in termini di score f1 ma scarso richiamo sulla classe positiva 1.

XGBoost

L'ultimo modello preso in esame per realizzare un'analisi di credit scoring è l'XGBoost, ovvero un algoritmo di apprendimento ensemble, conosciuto soprattutto per la sua velocità e accuratezza nelle sue predizioni, nel quale i nuovi modelli deboli vengono aggiunti per correggere gli errori commessi dai precedenti.

(target *anno*)

La fase di features selection produce come negli altri modelli il seguente dataset:

```
X
```

	Utile corrente/ricavi	Risultato netto rettif/AN	ROA	ROE	...	Ebitda-servizio debito/AN	Ln(AN)	Ln(RIC)
0	-8.65	-6.82	-2.65	-19.40		-5.34	11.10	11.11
1	1.43	0.64	2.08	1.82		-0.71	11.10	11.11
2	-6.01	-0.48	-3.32	-1.50		-2.27	11.10	11.11
3	1.54	-0.30	3.57	-1.00		-0.16	11.10	11.11
4	2.48	0.99	4.80	3.47		2.58	11.10	11.11
...	...	...	...	...		...	...	...
31831	0.00	0.00	3.23	-200.00		-18.58	4.39	1.10
31832	0.00	0.00	12.50	-200.00		-25.23	4.39	1.10
31833	0.00	0.00	11.11	0.00		-25.23	4.39	1.10
31834	-33.33	-11.11	0.00	-12.50		-25.23	4.39	1.10
31835	0.00	0.00	0.85	0.00		-4.88	4.77	1.10

31836 rows × 32 columns

```
y #y_anno
```

	FLAG di sana in liquid + anomala
0	0
1	0
2	0
3	0
4	0
...	...
31831	0
31832	0
31833	0
31834	0
31835	0

31836 rows × 1 columns

Dato che gli alberi decisionali non richiedono lo scaling dei dati di input, anche l'XGBoost non richiederà la normalizzazione del dataset tra 0 e 1 in quanto metodo d'insieme composto da n alberi decisionali. Si esegue esclusivamente lo split in set di train e test:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123456789)

# describes info about train and test set
print("X_train dataset: ", X_train.shape)
print("y_train dataset: ", y_train.shape)
print("X_test dataset: ", X_test.shape)
print("y_test dataset: ", y_test.shape)
```

```
X_train dataset:  (25468, 32)
y_train dataset:  (25468, 1)
X_test dataset:  (6368, 32)
y_test dataset:  (6368, 1)
```

Sebbene la libreria XGBoost abbia la propria API Python, si può richiamare il modello XGBoost con l'API scikit-learn utilizzando la classe wrapper *XGBClassifier*:

```
from xgboost import XGBClassifier

model = XGBClassifier(nthread= 4, objective ='binary:logistic',
                      scale_pos_weight=192, booster= 'gbtree')
```

Viene istanziato il modello, chiamato *model*, impostando inizialmente solo i parametri generali, come:

- *nthread*: numero di core (alberi/modelli base) paralleli utilizzati per eseguire l'XGBoost;
- *objective*: la funzione di costo da minimizzare; nel caso in esame è settata a 'binary:logistic' (regressione logistica per la classificazione binaria, con output una probabilità);
- *booster*: il booster da utilizzare; il valore di default è 'gbtree' quindi l'albero decisionale;
- *scale_pos_weight*: rapporto tra il numero di osservazioni appartenenti alla classe negativa (0) e quelle appartenenti alla classe positiva (1).

Come esposto nei modelli precedenti, utilizzando come risposta la target *anno* il dataset presenterà un forte sbilanciamento nelle due classi. Lo scale_pos_weight è stato progettato al fine di regolare il comportamento dell'algoritmo per problemi di classificazione sbilanciati. Un valore del scale_pos_weight più alto aiuta il modello a ottenere prestazioni migliori in sede di previsioni sulla classe positiva; tuttavia, se spinto troppo oltre, il modello rischia di sovrastimare la classe positiva a scapito di prestazioni peggiori sulla classe negativa o su entrambe le classi[27]. Un calcolo rapido per determinarne il valore da inserire per rendere bilanciato il dataset è:

$$scale\_pos\_weight = \frac{\#ossevazioni\ negative\ (train\ set)}{\#osservazioni\ positive\ (train\ set)}$$

Avendo nel set di addestramento un numero di esempi appartenenti alla classe 0 e 1 rispettivamente di 25336 e 132, lo scale_pos_weight da impostare è: $^{25336}/_{132} = 191.94 \approx 192$; questo implica il fatto che l'algoritmo assegnerà agli errori di classificazione commessi sulla classe 1 (minoranza) un impatto

[27] https://machinelearningmastery.com/xgboost-for-imbalanced-classification/

192 volte maggiore e, conseguentemente, una correzione 192 volte superiore rispetto agli errori sulla classe 0 (maggioranza).

Dunque, il modello iniziale è il seguente:

```
model
```

```
XGBClassifier(base_score=None, booster='gbtree', colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=None, gamma=None,
              gpu_id=None, importance_type='gain', interaction_constraints=None,
              learning_rate=None, max_delta_step=None, max_depth=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, nthread=4, num_parallel_tree=None,
              random_state=None, reg_alpha=None, reg_lambda=None,
              scale_pos_weight=192, subsample=None, tree_method=None,
              validate_parameters=None, verbosity=None)
```

Questo è l'insieme di tutti gli iperparametri che compone un modello XGBoost; essendo alcuni determinanti ai fini del risultato finale, è necessaria un'accurata fase di tuning per estrarne le migliori performance:

```
from sklearn.model_selection import RandomizedSearchCV

space = dict()
space['max_depth'] = [3, 4, 5, 6]
space['learning_rate'] = [0.001, 0.01, 0.05, 0.1]
space['gamma'] = [0, 0.5, 1, 5, 10]
space['min_child_weight'] = [4, 6, 7, 8]
space['subsample'] = [0.5, 0.7, 1]
space['colsample_bytree'] = [0.3, 0.4, 0.5, 0.7, 1]
space['n_estimators'] = [100, 150, 200]

grid = RandomizedSearchCV(model, space, scoring='recall', n_jobs=-1, cv=3)

random_result = grid.fit(X_train, y_train)
```

Viene creato lo spazio di ricerca nel quale si inseriscono gli iperparametri che si è deciso di ottimizzare con la Random Search:

- *max_depth* (default=6): profondità massima che può raggiungere ogni albero (modello debole); esso è utilizzato per controllare l'overfitting in quanto un'alta profondità consente al modello di apprendere relazioni molto specifiche (anche eccessive) per un particolare campione.

- *learning_rate* (default=0.3): nell'XGBoost vengono create nuovi alberi base per correggere gli pseudo-errori di previsione residui dal modello d'insieme corrente (sequenza di alberi corrente). Durante tale addestramento può accadere che il modello corrente si adatti eccessivamente ai dati di training e che quindi lo sovradimensioni. Una tecnica per rallentare l'apprendimento consiste nell'applicare un fattore di ponderazione per le correzioni dei nuovi alberi quando vengono aggiunti al modello; tale ponderazione è detta *shrinkage factor* o learning rate. Impostare valori bassi (come ad esempio, 0.0001 o 0.001) ha l'effetto di apportare meno correzioni per ogni albero aggiunto al modello; questo si traduce nell'aggiunta di più alberi al modello. In generale, si preferisce avere un lr tra 0.001 e 0.1[28].
- *gamma* (default=0): un nodo viene splittato sulla base del guadagno informativo derivante da tale split, ovvero quando la divisione fornisce una certa riduzione della funzione di costo; gamma rappresenta la riduzione minima della funzione di costo richiesta per eseguire lo split. Il valore impostato di default è 0, quindi lo split viene effettuato ogni volta che si ha un guadagno informativo positivo (anche se molto basso). Dato che splittare un nodo implica inevitabilmente aumentare la profondità dell'albero, troppe divisioni potrebbero portare l'albero ad adattarsi eccessivamente ai dati correnti, rischiando l'overfitting; aumentare il gamma rende l'algoritmo maggiormente conservativo[29].

[28] Se ad esempio, effettuando la Random Search uscisse un lr= 0.3, molto probabilmente il numero di alberi sarebbe troppo basso.
https://machinelearningmastery.com/tune-learning-rate-for-gradient-boosting-with-xgboost-in-python.

[29] Riduce la possibilità di split di ogni nodo in quanto si richiede un guadagno informativo superiore a una certa soglia minima (e che quindi la funzione di costo si riduca di almeno una certa quantità).

- *min_child_weight* (default=1): somma minima dei pesi di tutte le osservazioni richieste in un "bambino" (nodo creato successivamente a uno split de nodo "genitore"). Il valore di default è Anche questo iperparametro è utilizzato per controllare l'overfitting, infatti un valore più alto impedisce al modello di apprendere relazioni che potrebbero essere altamente specifiche per il particolare campione selezionato per un albero. Dall'altro lato, valori eccessivamente alti conducono all'underfitting.
- *subsample* (default=1): frazione di osservazioni da campionare in modo casuale per ogni albero.
- *colsample_bytree* (default=1): frazione di features (colonne) da campionare in modo casuale per ogni albero.
- *n_estimators*: numero di alberi (modelli deboli) da aggiungere sequenzialmente.

I risultati del tuning sono i seguenti:

```
Best: 0.757576 using {'subsample': 0.7, 'n_estimators': 100, 'min_child_weight': 4, 'max_depth': 3,
'learning_rate': 0.01, 'gamma': 1, 'colsample_bytree': 0.3}
```

La Random Search ha impostato dei parametri con dei valori che vanno a diminuire molto il rischio di overfitting del modello: ogni albero ha una bassa profondità massima ed è addestrato con il 30% delle features; inoltre, sia il learning_rate, il min_child _weight che il gamma presentano valori in linea con la riduzione dell'adattamento eccessivo ai dati correnti. I valori sono dunque ragionevoli.

A questo punto si procede con la validazione incrociata di questo modello tramite la 10-folds:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

model = XGBClassifier(nthread= 4, objective= 'binary:logistic', max_depth= 3, learning_rate= 0.01,
                      min_child_weight= 4, scale_pos_weight=192, booster= 'gbtree',
                      subsample= 0.7, gamma= 1, colsample_bytree= 0.3, n_estimators = 100)

scores = cross_val_score(model, X_train, y_train, cv=10)

for fold,score in enumerate(scores):
    print("Fold %d score=%.4f" % (fold+1,score))

print("\nValidation Accuracy = %.2f" % scores.mean())
```

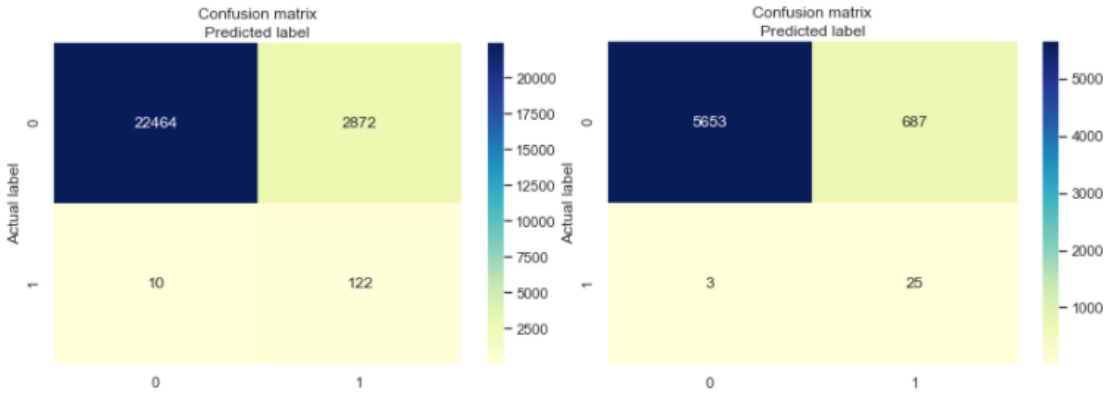
```
Fold 1 score=0.8869
Fold 2 score=0.8948
Fold 3 score=0.8956
Fold 4 score=0.8779
Fold 5 score=0.8857
Fold 6 score=0.8987
Fold 7 score=0.8956
Fold 8 score=0.8905
Fold 9 score=0.8755
Fold 10 score=0.8704

Validation Accuracy = 0.89
```

Essendo le accuracy si 10 folds estremamente vicine tra loro, il modello risulta convalidato. Successivamente, si ripete l'addestramento sull'intero training set:

```
model.fit(X_train, y_train)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.3, gamma=1, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.01, max_delta_step=0, max_depth=3,
              min_child_weight=4, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=4, nthread=4, num_parallel_tree=1,
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=192,
              subsample=0.7, tree_method='exact', validate_parameters=1,
              verbosity=None)
```

Non rimane che effettuare le predizioni su train e test ed osservare le performance ottenute:




```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.89	0.94	25336
1	0.04	0.92	0.08	132
accuracy			0.89	25468
macro avg	0.52	0.91	0.51	25468
weighted avg	0.99	0.89	0.94	25468

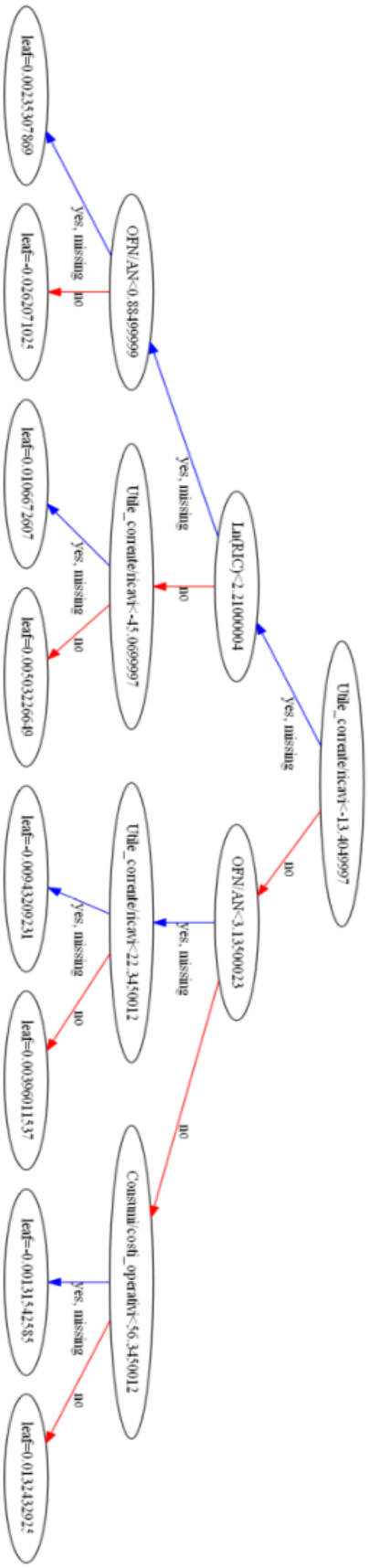
```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.89	0.94	6340
1	0.04	0.89	0.07	28
accuracy			0.89	6368
macro avg	0.52	0.89	0.51	6368
weighted avg	1.00	0.89	0.94	6368

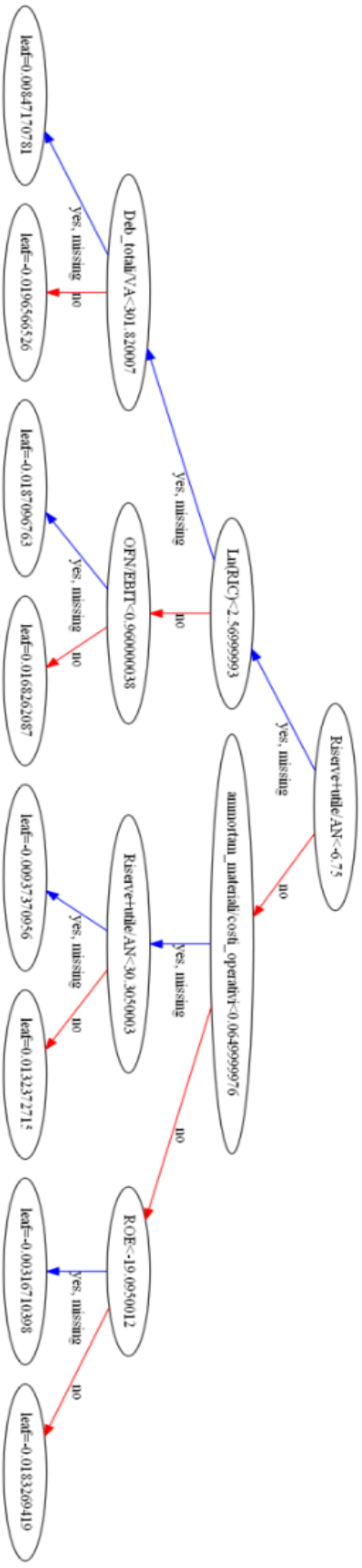
Il modello XGBoost in questione presenta risultati decisamente performanti; come in tutti gli altri modelli con target *anno*, la precisione rimane sbilanciata a favore della classe negativa a causa della bassa presenza di istanze di classe 1. Tuttavia, la principale caratteristica di questo modello è la seguente: gli ottimi risultati ottenuti sulla recall della classe 1 sono replicati sulla classe 0; in altre parole, l'aver raggiunto il 92% e l'89% rispettivamente nel train e test set non ha minimamente inficiato le performance sulla classe negativa, la quale è rimasta invariata sull'89%. La rete neurale sviluppata precedentemente non possiede questa particolarità in quanto, per arrivare a ottenere quelle determinate performance sulla recall della classe anomala, ha dovuto inevitabilmente aumentare gli errori commessi sull'altra classe, producendo uno squilibrio di circa il 12% tra di esse. Complessivamente, tale equilibrio tra le classi porta l'XGBoost a classificare correttamente un'osservazione l'89% delle volte.

Di seguito viene riportato l'albero iniziale (1°) e finale (100°) della sequenza:

Albero Finale



Albero Iniziale



(target *società*)

L'ultimo modello sviluppato è l'XGBoost con il dataset composto dalla target *società*; si ricorda che all'interno della variabile splittata *y_train* sono presenti:

```
y_train['FLAG di sana in liquid + anomala (società)'].value_counts()
```

```
0    21561
1     3907
Name: FLAG di sana in liquid + anomala (società), dtype: int64
```

viene, dunque, reimpostato il parametro *scale_pos_weight* a $^{21561}/_{3907} = 55.5185 \cong 5.52$.

```
from xgboost import XGBClassifier

model = XGBClassifier(nthread= 4, objective ='binary:logistic',
                      scale_pos_weight=5.52, booster= 'gbtree')
```

```
model
```

```
XGBClassifier(base_score=None, booster='gbtree', colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=None, gamma=None,
              gpu_id=None, importance_type='gain', interaction_constraints=None,
              learning_rate=None, max_delta_step=None, max_depth=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, nthread=4, num_parallel_tree=None,
              random_state=None, reg_alpha=None, reg_lambda=None,
              scale_pos_weight=5.52, subsample=None, tree_method=None,
              validate_parameters=None, verbosity=None)
```

Attraverso i medesimi processi di tuning degli iperparametri e di validazione incrociata, si ottengono i seguenti risultati:

```
Best: 0.741999 using {'subsample': 1, 'n_estimators': 150, 'min_child_weight': 6, 'max_depth': 4, 'le
arning_rate': 0.05, 'gamma': 10, 'colsample_bytree': 0.4}
```

```
Fold 1 score=0.8410
Fold 2 score=0.8347
Fold 3 score=0.8386
Fold 4 score=0.8312
Fold 5 score=0.8300
Fold 6 score=0.8296
Fold 7 score=0.8269
Fold 8 score=0.8253
Fold 9 score=0.8162
Fold 10 score=0.8425

Validation Accuracy = 0.83
```

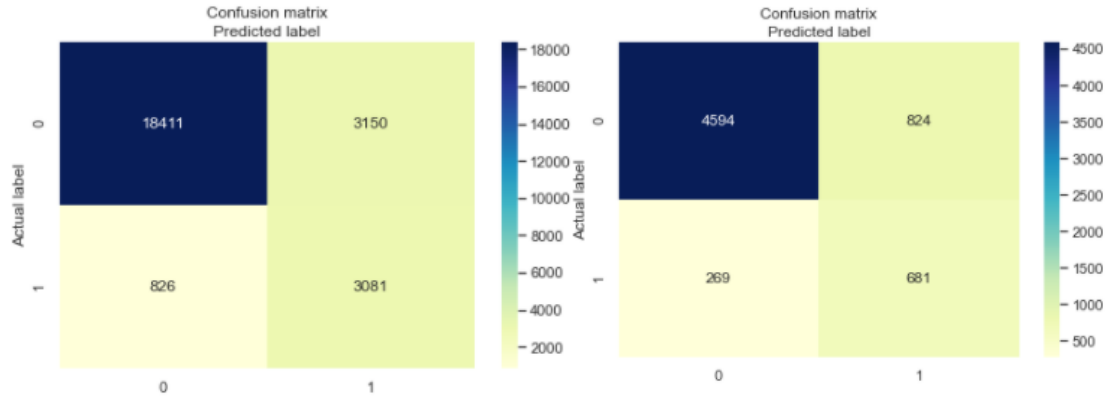
La Random Grid ha selezionato, anche in questo caso, dei valori per i vari iperparametri conservativi, al fine di ovviare al problema dell'overfitting.

Il modello da addestrare sarà quindi:

```
model.fit(X_train, y_train)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.4, gamma=10, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.05, max_delta_step=0, max_depth=4,
              min_child_weight=6, missing=nan, monotone_constraints='()',
              n_estimators=150, n_jobs=4, nthread=4, num_parallel_tree=1,
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=5.52,
              subsample=1, tree_method='exact', validate_parameters=1,
              verbosity=None)
```

Di seguito i risultati delle predizioni:




```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	0.96	0.85	0.90	21561
1	0.49	0.79	0.61	3907
accuracy			0.84	25468
macro avg	0.73	0.82	0.76	25468
weighted avg	0.89	0.84	0.86	25468

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.94	0.85	0.89	5418
1	0.45	0.72	0.55	950
accuracy			0.83	6368
macro avg	0.70	0.78	0.72	6368
weighted avg	0.87	0.83	0.84	6368

Come era da aspettarsi, anche utilizzando una metodologia profondamente diversa (metodo ensemble) la target società performa decisamente peggio della *anno*; tuttavia complessivamente, a differenza di quanto riscontrato nei modelli sviluppati in precedenza con la stessa target, il modello XGBoost offre prestazioni leggermente migliori, soprattutto in termini di equilibrio tra precisione e recall (f1-score).

Confronto

Alla luce dei risultati ottenuti, l'XGBoost risulta il modello maggiormente prestante e robusto per la tipologia di dataset che si ha a disposizione. Anche in molti altri casi pratici, dal confronto tra Regressione Logistica, Rete Neurale e XGBoost, è stata riscontrata questa particolare evidenza, ovvero il fatto che i modelli basati sugli alberi decisionali offrono prestazioni normalmente superiori agli altri nel momento in cui vengono trattati dataset strutturati (tabulari numerici o categorici). L'XGBoost, in particolare, è un modello ensemble strutturato su una *combinazione di n alberi decisionali*, che è il suo reale valore aggiunto. Se si guardano le performance esclusivamente sulla classe delle anomale e, quindi, della capacità di ridurre l'errore di secondo tipo (società anomala che il modello considera sana), la Regressione Logistica risulta, al tempo stesso, il modello meno prestante ma rispetto al quale si riesce a gestire totalmente l'entrata delle varie features nel modo corretto in termini sia di segno sia di significatività; tuttavia, questo significa avere una fase di features selection che, andando a rimuovere un gran numero di variabili (e quindi di dati), rischia di scaturire l'underfitting. Al contrario, la gestione di come entrano le features è una caratteristica che manca nella Rete Neurale e nell'XGBoost, in quanto sono modelli che ignorano completamente i problemi di segno e di stabilità.

In definitiva, se l'obiettivo è quello di avere il *pieno controllo sulle features* allora la scelta ricade sulla Regressione Logistica; altrimenti, qualora l'obiettivo fosse la *performance*, l'XGBoost (oppure gli altri modelli ensemble) sono certamente da preferire. La Rete Neurale viene utilizzata normalmente per altri tipi di dataset, dunque per quelli non strutturati (ad esempio contenenti immagini).

Analisi dell'Errore

Dopo aver concluso lo sviluppo di un modello di Machine Learning è necessario, come ultimo step, effettuare un'analisi degli errori di predizione commessi. L'esito di ogni singola predizione può essere:

- ✓ Anomala corretta: osservazione anomala che il modello considera correttamente anomala (predizione = 1, reale = 1).
- × Anomala errata: osservazione anomala che il modello considera erroneamente sana (predizione = 0, reale = 1).
- × Sana errata: osservazione sana che il modello considera erroneamente anomala (predizione = 1, reale = 0).
- ✓ Sana corretta: osservazione sana che il modello considera correttamente sana (predizione = 0, reale = 0).

In altre parole, ogni modelli predice correttamente una parte delle osservazioni sane e una parte delle osservazioni anomale e commette degli errori. Tali errori derivano dal fatto che alcune osservazioni, pur avendo un profilo finanziario più vicino alle sane (e che quindi il modello considera sane), risultano in realtà anomale (anomale errate); allo stesso modo, sono presenti delle osservazioni sane che, dato il profilo finanziario vicino a quello delle anomale, il modello le classifica come tali (sane errate).

L'analisi dell'errore consiste nell'individuare, se esistono (ma non è detto che esistano) delle features rispetto alle quali il profilo delle anomale errate, anziché essere più vicino alle sane, risulta maggiormente vicino alle anomale corrette e, contemporaneamente, se esistono delle variabili dove il profilo delle sane errate è più vicino a quello delle sane corrette invece che a quello delle anomale. Se tali variabili esistono, verranno inserite nel modello al fine di

tentare di ridurre l'errore sulle sane errate e, soprattutto, sulle anomale errate (errore decisamente più grave).

Nella pratica, per ognuno dei modelli sviluppati (si prende in considerazione la Regressione Logistica Base, Rete Neurale e XGBoost, strutturate con la target *anno*) vengono divise tutte le osservazioni (sia di train che di test) nelle 4 sezioni rappresentanti i possibili esiti, per ogni osservazione si estrae il suo corrispondente valore numerico (relativo a una feature) e se ne effettua una media aritmetica; dunque, per una certa feature x si otterranno 4 medie:

- Media sulle osservazioni anomale corrette;
- Media sulle osservazioni anomale errate;
- Media sulle osservazioni sane errate;
- Media sulle osservazioni sane corrette.

Si ripete questo procedimento per tutte le features *non entranti del modello* (che sono rimaste fuori). Infine, calcolando un *test t di Student per la differenza tra due medie*, si va a controllare se le medie sulle quattro sezioni (prese 2 a 2) sono statisticamente diverse oppure se non si può rifiutare il fatto che siano uguali.

Regressione Logistica – Base

Ponendo l'attenzione sul modello logit, le variabili che lo hanno composto sono state le seguenti: Ricavi/AN, AC/PC, Riserve+utile/AN, Debiti finanziari/EBITDA, Ln(AN).

Si procede, quindi, al calcolo, per tutte le altre variabili, delle medie sulle 4 sezioni:

	MEDIE			
FEATURES	CORRETTE ANOMALE	ANOMALE ERRATE	SANE ERRATE	CORRETTE SANE
EBITDA/Ricavi	-32,894	3,650	-14,719	9,417
EBIT/ricavi	-50,815	-9,099	-26,254	4,570
Utile corrente/ricavi	-58,870	-12,836	-31,105	3,363
Risultato netto rettif/ricavi	-51,976	-8,419	-26,982	1,634
Risultato netto rettif/AN	-9,154	0,689	-4,202	2,556
EBITDA/AN	-4,324	7,360	0,095	8,980
ROA	-7,981	3,587	-2,519	5,468
ROE	-167,248	-40,350	-82,514	5,516
ROE ante imposte	-166,125	-32,046	-80,024	15,731
Val Agg Oper/Ricavi	-10,537	21,870	5,232	28,200
Consumi/costi operativi	28,867	35,036	34,095	47,548
servizi esterni/costi operativi	47,976	42,285	45,496	28,591
costo lavoro/costi operativi	15,993	16,565	13,618	18,827
ammortam materiali/costi operativi	2,672	2,998	2,472	3,458
Val Agg/ITN	241,268	776,128	240,124	338,141
gg magazz	48,477	36,036	57,492	58,149
AC-mag/PC	88,790	170,412	134,221	140,202
Liq/PC	23,393	51,825	36,867	35,072
Liq/AN	14,540	15,166	11,189	9,888
Cap Circ/AN	-25,004	16,455	1,696	18,750
Patr netto/AN	-9,212	36,565	16,311	32,873
Patr netto tang/AN	-18,473	34,036	10,203	30,492
Patr netto/debiti totali	14,434	169,243	81,372	105,180
Patr netto tan/Debiti tot+PN	-18,667	37,370	12,688	33,218
Patr netto tan/Debiti tot-Liq+PN	-12,313	47,907	20,130	40,897
OF/RIC	7,634	4,935	6,685	1,684
OFN/RIC	5,717	3,859	4,447	1,313
OFN/EBITDA	345,877	122,449	256,867	29,150
OFN/EBIT	358,643	230,130	286,078	73,192
OFN/AN	1,781	1,321	1,221	0,943
OFN/Autof lordo	330,190	144,908	248,959	42,141
Autof Lordo/AN (=cash flow/attivo)	-4,146	6,013	-0,370	7,358
Auto Lordo-comp straord/AN	-5,467	5,250	-0,980	7,311
Deb totali/VA	675,243	456,057	587,652	341,487
Deb totali/Ric	1039,690	446,584	843,434	116,303
PC/ric	578,434	263,165	424,229	76,349
Deb finanziari (stimati)/VA	458,928	307,536	377,721	177,578
Deb finanziari (stimati)/Ric	396,282	216,962	378,495	62,039
Debiti totali/EBITDA	841,697	262,664	624,275	41,441
Ebitda-servizio debito/AN	-17,395	0,643	-9,270	3,988
Ln(RIC)	5,627	6,472	5,222	7,819

E al calcolo del t test:

	A	B	C	D	E	F
1		t test differenza tra 2 medie				
2	FEATURES	AN ERR & SA ERR	CORR AN & AN ERR	CORR AN & SA ERR	CORR SA & SA ERR	CORR SA & AN ERR
3	EBITDA/Ricavi	**3,808**	-6,439	-5,928	50,183	1,201
4	EBIT/ricavi	**2,740**	-5,762	-6,588	52,102	2,192
5	Utile corrente/ricavi	**2,760**	-6,089	-7,393	55,844	2,458
6	Risultato netto rettif/ricavi	**3,276**	-6,518	-6,889	51,745	1,783
7	Risultato netto rettif/AN	**2,487**	-4,633	-6,042	57,523	0,951
8	EBITDA/AN	**3,897**	-5,856	-6,056	71,450	0,870
9	ROA	**3,237**	-5,708	-7,174	62,850	0,999
10	ROE	**2,536**	-7,153	-13,086	65,278	2,768
11	ROE ante imposte	**2,647**	-6,963	-12,679	68,185	2,643
12	Val Agg Oper/Ricavi	**3,391**	-5,656	-5,182	44,859	1,296
13	Consumi/costi operativi	**0,198**	-1,193	-2,500	34,654	2,638
14	servizi esterni/costi operativi	**-0,775**	1,219	1,120	-43,406	-3,318
15	costo lavoro/costi operativi	**1,091**	-0,191	1,786	22,692	0,840
16	ammortam materiali/costi operativi	**0,706**	-0,393	0,530	18,048	0,619
17	Val Agg/ITN	**3,104**	-2,948	0,020	11,841	-2,539
18	gg magazz	**-1,953**	0,968	-1,317	0,584	2,022
19	AC-mag/PC	**1,298**	-2,747	-4,248	2,751	-1,086
20	Liq/PC	**1,045**	-1,911	-3,130	-1,734	-1,173
21	Liq/AN	**1,339**	-0,191	2,326	-5,598	-1,781
22	Cap Circ/AN	**2,227**	-5,582	-7,759	30,068	0,347
23	Patr netto/AN	**3,442**	-7,241	-10,631	34,280	-0,629
24	Patr netto tang/AN	**3,825**	-7,720	-10,168	38,114	-0,570
25	Patr netto/debiti totali	**2,342**	-4,007	-6,829	9,493	-1,711
26	Patr netto tan/Debiti tot+PN	**3,601**	-7,272	-8,668	33,449	-0,608
27	Patr netto tan/Debiti tot-Liq+PN	**3,009**	-5,775	-6,527	25,939	-0,762
28	OF/RIC	**-1,392**	1,869	1,299	-41,364	-2,598
29	OFN/RIC	**-0,647**	1,796	2,502	-38,919	-2,813
30	OFN/EBITDA	**-4,537**	7,067	7,681	-89,317	-3,160
31	OFN/EBIT	**-1,670**	3,688	7,128	-85,779	-4,696
32	OFN/AN	**0,361**	1,470	3,823	-13,326	-1,366
33	OFN/Autof lordo	**-3,118**	5,181	6,078	-78,455	-3,088
34	Autof Lordo/AN (=cash flow/attivo)	**3,211**	-4,719	-4,489	66,111	0,678
35	Auto Lordo-comp straord/AN	**3,714**	-5,970	-6,828	73,547	1,231
36	Deb totali/VA	**-1,502**	2,138	1,607	-29,826	-1,313
37	Deb totali/Ric	**-2,535**	3,219	1,969	-45,942	-2,120
38	PC/ric	**-1,953**	3,310	3,159	-46,119	-2,274
39	Deb finanziari (stimati)/VA	**-1,104**	2,156	2,653	-38,075	-2,050
40	Deb finanziari (stimati)/Ric	**-2,398**	2,282	0,428	-45,617	-2,311
41	Debiti totali/EBITDA	**-4,702**	7,005	6,857	-88,086	-2,887
42	Ebitda-servizio debito/AN	**5,118**	-8,383	-8,410	76,252	1,733
43	Ln(RIC)	**2,652**	-1,629	1,789	55,736	2,871

In dettaglio, il t test è tra:

- Colonna B: media anomale errate & media sane errate;
- Colonna C: media anomale corrette & media anomale errate;
- Colonna D: media anomale corrette & media sane errate;

- Colonna E: media sane corrette & media sane errate;
- Colonna F: media sane corrette & media anomale errate;

Riprendendo quanto detto in precedenza, affinché una features esterna possa essere inserita nel modello è necessario che:

1) il t tra le medie (anomale errate & sane errate), (anomale errate & anomale corrette) e (sane errate & sane corrette) risulti *non statisticamente significativo* per un α=5%, in modo tale da *non poter rifiutare l'ipotesi nulla H_0 che le due medie siano uguali*:

$$|t|_{col\ B} < 1.96$$

$$|t|_{col\ C} < 1.96$$

$$|t|_{col\ E} < 1.96$$

2) il profilo delle *anomale errate* sia più vicino a quello delle anomale corrette rispetto a quello delle sane corrette:

$$|t|_{col\ C} < |t|_{col\ F}$$

3) il profilo delle *sane errate* sia più vicino a quello delle sane corrette rispetto a quello delle anomale corrette:

$$|t|_{col\ E} < |t|_{col\ D}$$

Da questa serie di condizioni, le quali devono essere rispettate contemporaneamente, l'unica feature che risulta inseribile nel modello è *gg_magazz*. Tuttavia, questa variabile era stata rimossa a causa della sua *non*

significatività: al momento della rimozione presentava un p value del 34%, dunque non si poteva rifiutare l'ipotesi nulla H_0 che fosse statisticamente uguale a zero.

Provando a inserirla ugualmente nel modello:

Senza Inserimento

```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.84	0.91	25336
1	0.02	0.78	0.05	132
accuracy			0.84	25468
macro avg	0.51	0.81	0.48	25468
weighted avg	0.99	0.84	0.91	25468

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.84	0.91	6340
1	0.02	0.89	0.05	28
accuracy			0.84	6368
macro avg	0.51	0.87	0.48	6368
weighted avg	1.00	0.84	0.91	6368

Con l'Inserimento

```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.84	0.91	25336
1	0.02	0.77	0.05	132
accuracy			0.84	25468
macro avg	0.51	0.80	0.48	25468
weighted avg	0.99	0.84	0.91	25468

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.84	0.91	6340
1	0.02	0.86	0.05	28
accuracy			0.84	6368
macro avg	0.51	0.85	0.48	6368
weighted avg	0.99	0.84	0.91	6368

Si osserva un lieve peggioramento, sia sul train che sul test set, rispetto al caso originale in termini di recall sulla classe delle anomale e un'accuracy generale che rimane costante all'84%.

Dunque, si può affermare che essendo la feature *gg_magazz* rappresentante di una fetta dell'Attivo Netto (AN) e calcolata anche attraverso i Ricavi:

$$gg\_magazz = \frac{magazzino}{Ricavi} \cdot 360$$

è molto probabile che la variabile Ricavi/AN (*interna* al modello) possa aver già catturato la sua informazione in chiave multivariata.

Rete Neurale Artificiale

FEATURES	MEDIE			
	CORRETTE ANOMALE	ANOMALE ERRATE	SANE ERRATE	CORRETTE SANE
EBITDA/Ricavi	-28,797	14,024	-19,554	10,198
EBIT/ricavi	-46,744	10,221	-32,502	5,789
Risultato netto rettif/ricavi	-47,611	10,334	-33,608	2,982
EBITDA/AN	-3,312	14,356	-2,165	9,325
ROE ante imposte	-153,028	29,889	-102,725	19,913
AC-mag/PC	93,065	253,718	137,489	138,280
Patr netto/AN	-3,345	40,504	15,911	32,932
Patr netto tan/Debiti tot+PN	-11,443	41,667	11,985	33,390
Patr netto tan/Debiti tot-Liq+PN	-4,724	54,678	19,600	40,877
OF/RIC	7,606	0,782	7,625	1,341
OFN/EBITDA	325,352	3,205	323,453	21,429
Autof Lordo/AN (=cash flow/attivo)	-3,446	14,320	-2,611	7,737
Deb totali/Ric	991,868	47,874	869,719	90,439
Debiti totali/EBITDA	784,263	5,964	775,031	27,810

	A	B	C	D	E	F
1		t test differenza tra 2 medie				
2	FEATURES	AN ERR & SA ERR	CORR AN & AN ERR	CORR AN & SA ERR	CORR SA & SA ERR	CORR SA & AN ERR
3	EBITDA/Ricavi	**10,716**	-10,058	-3,141	70,257	-1,232
4	EBIT/ricavi	**15,145**	-12,758	-4,026	73,936	-1,598
5	Risultato netto rettif/ricavi	**21,433**	-14,706	-4,078	75,528	-3,689
6	EBITDA/AN	**7,210**	-7,363	-1,586	117,757	-2,197
7	ROE ante imposte	**5,883**	-7,759	-7,097	105,841	-0,443
8	AC-mag/PC	**2,469**	-3,341	-4,388	0,387	-2,454
9	Patr netto/AN	**2,875**	-4,911	-7,308	37,561	-0,886
10	Patr netto tan/Debiti tot+PN	**2,802**	-4,747	-6,373	37,661	-0,782
11	Patr netto tan/Debiti tot-Liq+PN	**2,406**	-3,874	-5,011	28,460	-0,947
12	OF/RIC	**-21,637**	9,357	-0,028	-56,134	1,887
13	OFN/EBITDA	**-105,906**	26,561	0,158	-166,648	7,392
14	Autof Lordo/AN (=cash flow/attivo)	**9,480**	-9,087	-1,038	113,053	-3,690
15	Deb totali/Ric	**-42,431**	10,252	1,325	-54,756	3,204
16	Debiti totali/EBITDA	**-146,360**	23,741	0,279	-146,831	12,781

In figura sono mostrate, come esposto in precedenza, le features che non sono entrate nel modello: sono di un numero molto inferiore rispetto alla Regressione Logistica in quanto per la Rete Neurale (così come per l'XGBoost) si è effettuata una fase di features selection molto ridotta. Eseguendo gli stessi controlli non viene individuata alcuna feature da poter inserire e quindi si afferma che il modello di Rete Neurale *non è migliorabile* mediante le variabili in figura.

XGBoost

FEATURES	MEDIE			
	CORRETTE ANOMALE	ANOMALE ERRATE	SANE ERRATE	CORRETTE SANE
EBITDA/Ricavi	-28,548	7,908	-29,266	8,513
EBIT/ricavi	-46,679	5,103	-45,130	3,637
Risultato netto rettif/ricavi	-47,560	5,300	-46,557	1,037
EBITDA/AN	-3,024	9,740	-3,511	8,370
ROE ante imposte	-153,156	17,258	-142,277	12,907
AC-mag/PC	99,339	170,414	121,783	140,191
Patr netto/AN	-3,111	34,480	5,278	32,611
Patr netto tan/Debiti tot+PN	-11,280	35,730	-1,247	32,968
Patr netto tan/Debiti tot-Liq+PN	-4,395	46,384	6,184	40,491
OF/RIC	7,630	1,034	8,967	1,787
OFN/EBITDA	320,927	78,023	337,679	49,211
Autof Lordo/AN (=cash flow/attivo)	-3,196	10,118	-4,133	6,916
Deb totali/Ric	998,362	47,061	1130,663	133,738
Debiti totali/EBITDA	775,896	160,444	831,904	93,800

	A	B	C	D	E	F
1		t test differenza tra 2 medie				
2	FEATURES	AN ERR & SA ERR	CORR AN & AN ERR	CORR AN & SA ERR	CORR SA & SA ERR	CORR SA & AN ERR
3	EBITDA/Ricavi	**16,759**	-9,957	0,237	63,275	0,283
4	EBIT/ricavi	**21,422**	-12,346	-0,428	67,300	-0,657
5	Risultato netto rettif/ricavi	**15,754**	-11,213	-0,286	70,155	-1,323
6	EBITDA/AN	**5,569**	-5,115	0,628	84,470	-0,576
7	ROE ante imposte	**7,602**	-7,705	-1,499	109,041	-0,208
8	AC-mag/PC	**1,464**	-2,040	-2,027	7,101	-0,912
9	Patr netto/AN	**4,519**	-5,390	-3,056	46,219	-0,290
10	Patr netto tan/Debiti tot+PN	**5,332**	-5,995	-2,630	44,128	-0,401
11	Patr netto tan/Debiti tot-Liq+PN	**4,000**	-4,554	-2,101	33,273	-0,589
12	OF/RIC	**-19,075**	8,523	-1,943	-46,231	1,946
13	OFN/EBITDA	**-6,290**	5,646	-1,338	-123,209	-0,699
14	Autof Lordo/AN (=cash flow/attivo)	**5,334**	-4,766	1,121	80,264	-1,200
15	Deb totali/Ric	**-51,523**	10,368	-1,412	-49,617	12,373
16	Debiti totali/EBITDA	**-6,486**	5,667	-1,653	-119,602	-0,645

Anche in questo caso, l'XGBoost *non risulta migliorabile*.

Modelli con l'aggiunta di Variabili Qualitative

L'analisi dell'Errore ha rimarcato l'impossibilità di migliorare ulteriormente le performance dei modelli attraverso esclusivamente le variabili quantitative (numeriche). Un ultimo tentativo è rappresentato dall'inserimento nei modelli delle variabili *qualitative* che si hanno attualmente a disposizione dal dataset importato da Aida:

- anno di riferimento del bilancio (dal 2009 al 2018)
- area geografica (Nord-Ovest, Nord-Est, Centro, Sud-Isole)
- forma giuridica (S.P.A., S.R.L., Cooperativa, S.A.S., S.N.C.)

Attraverso il metodo *one-hot-encoding* (eseguito con il comando Pandas "get_dummies"), si sostituisce la singola colonna (variabile) con più colonne, una per ogni label (valore distinto assunto dalla variabile), il cui contenuto sarà una variabile booleana 0/1 che indica l'appartenenza o meno alla classe rappresentata dal label. Ad esempio, considerando la variabile "area geografica", si passa da avere la singola feature:

	anno di riferimento del bilancio	Forma Giuridica	Zona
0	2009	S.P.A.	Nord-Ovest
1	2010	S.P.A.	Nord-Ovest
2	2011	S.P.A.	Nord-Ovest
3	2009	S.R.L.	Nord-Ovest
4	2010	S.R.L.	Nord-Ovest
...	...	...	...
31831	2009	S.R.L.	Centro
31832	2009	S.R.L.	Centro
31833	2009	S.R.L.	Sud-Isole
31834	2010	S.R.L.	Sud-Isole
31835	2009	S.R.L.	Nord-Ovest

ad avere 4 variabili, una per ogni valore distinto assunto dalla variabile originale:

	anno di riferimento del bilancio	Forma Giuridica	Zona_Centro	Zona_Nord-Est	Zona_Nord-Ovest	Zona_Sud-Isole
0	2009	S.P.A.	0	0	1	0
1	2010	S.P.A.	0	0	1	0
2	2011	S.P.A.	0	0	1	0
3	2009	S.R.L.	0	0	1	0
4	2010	S.R.L.	0	0	1	0
...	...	...	...	...	...	...
31831	2009	S.R.L.	1	0	0	0
31832	2009	S.R.L.	1	0	0	0
31833	2009	S.R.L.	0	0	0	1
31834	2010	S.R.L.	0	0	0	1
31835	2009	S.R.L.	0	0	1	0

Come si sottolinea, l'osservazione 0 essendo in zona Nord-Ovest avrà, nel nuovo dataset, l'1 in corrispondenza della variabile "Zona_Nord-Ovest". La medesima operazione si esegue sulle altre due variabili.

Passando direttamente ai risultati:

- **Regressione Logistica:**

Senza Dummies

```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.84	0.91	25336
1	0.02	0.78	0.05	132
accuracy			0.84	25468
macro avg	0.51	0.81	0.48	25468
weighted avg	0.99	0.84	0.91	25468

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.84	0.91	6340
1	0.02	0.89	0.05	28
accuracy			0.84	6368
macro avg	0.51	0.87	0.48	6368
weighted avg	1.00	0.84	0.91	6368

Con Dummies

```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.85	0.92	25336
1	0.03	0.79	0.05	132
accuracy			0.84	25468
macro avg	0.51	0.82	0.48	25468
weighted avg	0.99	0.84	0.91	25468

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.85	0.92	6340
1	0.02	0.82	0.05	28
accuracy			0.85	6368
macro avg	0.51	0.84	0.48	6368
weighted avg	0.99	0.85	0.92	6368

L'introduzione delle 3 variabili dummies ha portato a un modello sicuramente più equilibrato passando dal train al test set: infatti, la recall sulla classe 1,

invece che andare dal 78% all'89%, passa dal 79% all'82%; questo fatto da un lato indica una minore performance nel catturare correttamente le osservazioni anomale ma dall'altro riduce il problema di underfitting che affligge, al contrario, il modello senza dummies, il quale ottiene risultati forse eccessivamente migliori nel test rispetto che nel train set. L'accuratezza generale è rimasta pressoché costante.

- **Rete Neurale:**

Senza Dummies

```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.80	0.89	25336
1	0.02	0.92	0.05	132
accuracy			0.80	25468
macro avg	0.51	0.86	0.47	25468
weighted avg	0.99	0.80	0.88	25468

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.81	0.89	6340
1	0.02	0.93	0.04	28
accuracy			0.81	6368
macro avg	0.51	0.87	0.47	6368
weighted avg	1.00	0.81	0.89	6368

Con Dummies

```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.75	0.86	25336
1	0.02	0.93	0.04	132
accuracy			0.75	25468
macro avg	0.51	0.84	0.45	25468
weighted avg	0.99	0.75	0.85	25468

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.76	0.86	6340
1	0.02	0.93	0.03	28
accuracy			0.76	6368
macro avg	0.51	0.85	0.45	6368
weighted avg	1.00	0.76	0.86	6368

In questo caso il nuovo modello risultante è decisamente peggiore di quello privo di dummies in quanto, a parità di performance sulle recall (1), sono peggiorate di 5 punti percentuali le predizioni sulla classe delle sane e, conseguentemente, anche dell'accuracy totale.

- **XGBoost:**

Senza Dummies

```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.89	0.94	25336
1	0.04	0.92	0.08	132
accuracy			0.89	25468
macro avg	0.52	0.91	0.51	25468
weighted avg	0.99	0.89	0.94	25468

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.89	0.94	6340
1	0.04	0.89	0.07	28
accuracy			0.89	6368
macro avg	0.52	0.89	0.51	6368
weighted avg	1.00	0.89	0.94	6368

Con Dummies

```
print(classification_report(y_train, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.88	0.94	25336
1	0.04	0.94	0.08	132
accuracy			0.88	25468
macro avg	0.52	0.91	0.51	25468
weighted avg	0.99	0.88	0.93	25468

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.89	0.94	6340
1	0.03	0.89	0.07	28
accuracy			0.89	6368
macro avg	0.52	0.89	0.50	6368
weighted avg	1.00	0.89	0.94	6368

La situazione nel XGBoost è rimasta sostanzialmente invariata; da segnalare è il comportamento del modello con dummies nel training set in cui l'errore sul richiamo della classe positiva si è ridotto di 2 punti percentuali (classifica correttamente 2 osservazioni in più, riducendo le anomale errate da 10 -in quello senza dummies- a 8) a scapito di una lieve riduzione della performance nell'altra classe.

In definitiva, quelli presentati sono i migliori risultati che si sono potuti ottenere utilizzando i dati presenti su Aida. Per ulteriori miglioramenti si ritiene siano necessari, dunque, altri dati di natura qualitativa come ad esempio informazioni specifiche del settore, le sue caratteristiche peculiari, sui mercati oppure dati quantitativi meno inquinati.

CONCLUSIONE

Questo lavoro di tesi ha cercato di rispondere alla domanda: "Qual è il modello di Machine Learning più efficacie, in termini di performance e solidità, per lo svolgimento di un'analisi in merito creditizio?" A tal fine, sono state utilizzate e, infine, confrontate quelle tecniche che in letteratura sono riconosciute come tra le più attendibili, analizzandone i pregi e difetti.

Tutti i modelli sviluppati hanno portato a risultati discreti -in alcuni casi ottimi- e, come esaminato nell'ultima sezione del progetto, a performance in ogni caso non facilmente incrementabili se non con l'aggiunta di altre tipologie di variabili o informazioni supplementari. Questo conduce alla conclusione che i modelli proposti non solo risultano essere performanti e, al tempo stesso, robusti nelle loro predizioni ma dimostrano anche il fatto di aver utilizzato correttamente e razionalmente le informazioni e dati inizialmente a disposizione.

Come in moltissimi altri casi, quando si tratta di Machine Learning la risposta alla domanda iniziale è "*dipende*". Si ritiene che la preferenza tra uno piuttosto che un altro dipenda generalmente non dalla validità intrinseca del modello, in quanto ogni modello storicamente è stato pensato e sviluppato per gestire in un certo modo un determinato problema, ma dal risultato e dal "come" si vuole ottenere tale risultato. Questo non significa che non esistano in assoluto modelli migliori di altri ma per stabilirlo correttamente è necessario chiedersi in anticipo per quale scopo è nato tale modello e come è stato strutturato per raggiungerlo.

SITOGRAFIA

- Immagini e Teoria dei capitoli 2 (*Rischio di Credito: Regressione Logistica*), 3 (*Rete Neurale Artificiale*) e 5 (*Aspetti comuni dei modelli*) sono presi da: udemy.com/course/deep-learning-pratico/learn/lecture/14257618#overview
- https://www.schededigeografia.net/Italia/Economia/industrie_metallurgiche_siderurgiche_italia.htm
- https://www.britannica.com/science/metallurgy/Brass
- https://www.agi.it/fact-checking/dati_produzione_acciaio-6683455/news/2019-12-06/
- https://www.worldsteel.org/steel-by-topic/statistics/World-Steel-in-Figures.html (utilizzati tutti i PDF dal 2002 al 2020)
- https://www.oecd.org/sti/ind/43312347.pdf
- https://www.istat.it/it/archivio/247258
- https://www.istat.it/it/files//2020/09/foit202007.pdf
- http://federacciai.it/pubblicazioni-varie/
- https://lorenzogovoni.com/regressione-logistica/
- https://www.comarch.com/finance/articles/what-is-credit-scoring-about-types-model-and-method/
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8
- https://medium.com/datadriveninvestor/neural-networks-explained-6e21c70d7818
- https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
- https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/
- https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484

- https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205
- https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/
- https://randlow.github.io/posts/machine-learning/boosting-explain/
- https://www.mygreatlearning.com/blog/gradient-boosting/
- https://www.toptal.com/machine-learning/ensemble-methods-machine-learning
- https://blog.paperspace.com/gradient-boosting-for-classification/
- https://ichi.pro/it/algoritmi-di-potenziamento-adaboost-gradient-boosting-xgb-light-gbm-e-catboost-254892816262346
- https://machinelearningmastery.com/cost-sensitive-neural-network-for-imbalanced-classification/
- https://www.germanorossi.it/mi/file/psico/AF2.pdf
- https://www.datacamp.com/community/tutorials/introduction-factor-analysis
- https://machinelearningmastery.com/xgboost-for-imbalanced-classification/
- https://lorenzogovoni.com/come-utilizzare-lxgboost-in-python-per-problemi-di-regressione/
- https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/
- https://chrisalbon.com/deep_learning/keras/k-fold_cross-validating_neural_networks/
- https://machinelearningmastery.com/use-keras-deep-learning-models-scikit-learn-python/
- https://towardsdatascience.com/understanding-hyperparameters-and-its-optimisation-techniques-f0debba07568
- https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/
- https://blog.usejournal.com/a-comparison-of-grid-search-and-randomized-search-using-scikit-learn-29823179bc85
- https://xgboost.readthedocs.io/en/latest/parameter.html