# POLITECNICO DI TORINO

**Corso di Laurea Magistrale in Ingegneria Gestionale**

*Tesi di Laurea in Ingegneria Gestionale*

**Applicazione di modelli di Machine Learning ad Albero Decisionale con R per il Credit Scoring nel settore elettronico**


RELATORE
Prof. Franco Varetto

CANDIDATO
Alessandro Felice

# INTRODUZIONE

Quando si parla di Machine Learning ci si riferisce ai differenti meccanismi che permettono ad una macchina di migliorare le proprie capacità e prestazioni nel tempo.
Dagli anni cinquanta questa branca dell'informatica (che è una parte del più vasto campo dell'Intelligenza Artificiale) ha preso sempre più importanza. L'interesse derivante dall'apprendimento di una macchina capace poi di arrivare a formulare previsioni e decisioni sta avendo un forte impatto anche nella vita di tutti i giorni.
Lo spettro di progetti dove l'apprendimento automatico è il protagonista spazia da applicazioni scientifiche fino ad applicazioni quotidiane.
Si pensi all'enorme contributo che le tecniche di ML e AI stanno avendo nel mondo della sanità, per migliorare ad esempio il flusso di lavoro clinico, o nella ricerca, o ancora nella gestione del rischio finanziario.
Ad oggi si sono ottenuti già ottimi risultati ma le possibilità di sviluppo futuro di questo mondo sono ancora moltissime.

Il seguente lavoro di Tesi si concentra nell'analisi dell'utilizzo di tecniche di Machine Learning applicate alla rischiosità del credito. L'obiettivo è analizzare le funzionalità di certi modelli applicati a casistiche reali, evidenziandone le capacità nell'analisi e nell'elaborazione dati, anche tramite utilizzo di linguaggio di programmazione R.
Lo studio è rivolto in particolare ad algoritmi di decisione *tree-based,* quindi con l'utilizzo di alberi decisionali e con le varie applicazioni si restituisce un modello di previsione capace di classificare il rischio creditizio.
In particolare per l'analisi si è scelto di considerare le aziende appartenenti al settore elettronico, quindi interessate nella fabbricazione e nel retail di elettronica di consumo.

Nel dettaglio, nel primo capitolo, si analizzerà la definizione di rischio di credito, andando a vederne cause, componenti e modelli di valutazione del rischio.

Essendo un elaborato centrato soprattutto nella descrizione di algoritmi basati su alberi decisionali e della loro applicazione per analisi del rischio di credito sono stati introdotti concetti teorici utili per approfondirne le basi teoriche.

Nel secondo capitolo, insieme a modelli di valutazione del rischio di credito come il Modello Logit sarano infatti trattati modelli quali Regressione Ridge e LASSO, i cui fondamenti sono alla base del Boosting, e le matrici di classificazione che verranno riprese nei capitoli successivi per analizzare i risultati degli alberi decisionali.

Nel terzo capitolo saranno esposti nel dettaglio gli algoritmi basati su alberi decisionali, dal modello più semplice fino alle applicazioni più recenti, si partirà dall'algoritmo CART e si arriverà all'algoritmo XGBoost, passando per i modelli Bagging e quindi Boosting.
Saranno riportate le basi teoriche per capire al meglio le funzionalità e le potenzialità delle applicazioni degli stessi algoritmi utilizzati nel terzo capitolo.

L'ultima parte del lavoro di Tesi si concentrerà sulla scrittura del codice R degli algoritmi trattati, verranno esposti gli step dalla fase di preanalisi del dataset fino alla descrizione dell'output, con l'obiettivo di ottenere dei modelli decisionali accurati nelle previsioni per l'analisi del credito delle aziende prese in esame.

# 1. IL RISCHIO DI CREDITO

I mercati finanziari sono sempre più complessi e dinamici e le banche necessitano di migliorare la gestione dei propri processi per raggiungere la massima efficienza anche in ottica automazione, sempre in virtù della sempre maggior numerosità di strumenti finanziari.
Nel settore bancario è possibile individuare tre diverse tipologie di rischio:
- Rischi di mercato: rischi che influenzano l'andamento del mercato e che dipendono da fattori che non possono essere eliminati o ridotti anche mediante diversificazione del portafoglio. Le variabili di mercato che creano queste variazioni inattese sono ad esempio i prezzi, i tassi, i cambi e le relative volatilità.
- Rischi di credito: rischi in cui incorre il titolare di un'attività finanziaria per la variazione inattesa del merito di credito di una controparte che può risultare parzialmente o totalmente incapace ad assolvere l'impegno assunto.
- Rischi operativi: rischi derivanti da errori manuali, tecnici, malfunzionamento dei sistemi o interruzione di processi a causa di azioni di terzi.

Presteremo particolare attenzione al rischio di credito, dagli aspetti teorici generali arriveremo poi ad applicare modelli di Machine Learning quali alberi decisionali, Random Forest e Boosting.

## *1.1 Tipologie di rischio*

Con il termine Rischio di Credito si fa riferimento a diverse categorie di rischio applicabili a diverse tipologie di strumenti finanziari; è inteso come la possibilità di una variazione inattesa del merito creditizio di una controparte che possa poi generare una variazione inattesa della posizione creditizia.
In questa definizione si racchiudono concetti importanti quali:

• Il rischio di credito non è legato solo alla possibilità di insolvenza ma anche al deterioramento del merito creditizio
• Si parla propriamente di rischio di credito quando le variazioni della posizione sono inattese, proprio da determinate valutazioni e previsioni di quest'ultime si arriva poi alla gestione del finanziamento
• Il rischio di credito comprende variazioni applicabili non solo alle posizioni in bilancio ma anche a tutti quegli strumenti derivati di mercati cosiddetti Over The Counter.

Questa "natura" del credito si può poi scomporre in diverse tipologie a seconda delle cause che lo generano:

1. Rischio di insolvenza: rappresenta la possibilità che una controparte diventi insolvente, in tal caso c'è una perdita economica per il creditore
2. Rischio di migrazione: rischio di deterioramento del merito creditizio di una controparte. Introduciamo quindi il concetto di *rating* del debitore, un deterioramento del credito intacca il rating del debitore, che porta ad un declassamento di quest'ultimo
3. Rischio di recupero: è il rischio per il creditore legato all'ammontare effettivamente recuperato che può rivelarsi inferiore di quanto originariamente stimato, in seguito alla controparte divenuta insolvente. A tale rischio fa riferimento il concetto di Loss Given Default (perdita in caso di insolvenza) o più brevemente LGD
4. Rischio di esposizione: il rischio legato all'effettivo ammontare del prestito al momento dell'insolvenza, l'exposure at Default (EAD) può risultare anche molto differente da quella corrente
5. Rischio di spread: rischio legato allo spread dei mercati di capitali, indica la possibilità che, a parità di merito creditizio, aumenti il premio al rischio. E' un rischio che si verifica ad esempio in una crisi di liquidità o in funzione della considerazione che gli investitori hanno dei mercati, lo spread sale con titoli ritenuti rischiosi o inaffidabili.

## *1.2 Componenti del rischio di credito*

Introduciamo il concetto di Perdita attesa e Perdita inattesa.

La Perdita attesa è il valore medio della distribuzione delle perdite, come suggerisce il nome è una componente "misurabile", non può essere eliminata diversificando il portafoglio e viene già inclusa nello spread creditizio.
Si calcola prendendo in esame:

- l'esposizione in caso di insolvenza AE (Adjusted Exposure)
- probabilità di insolvenza PD
- la Loss Given Default LGD, la percentuale di perdita qualora si verificasse l'insolvenza della controparte
- Recovery Rate RR, il tasso di recupero del credito in caso di insolvenza della controparte

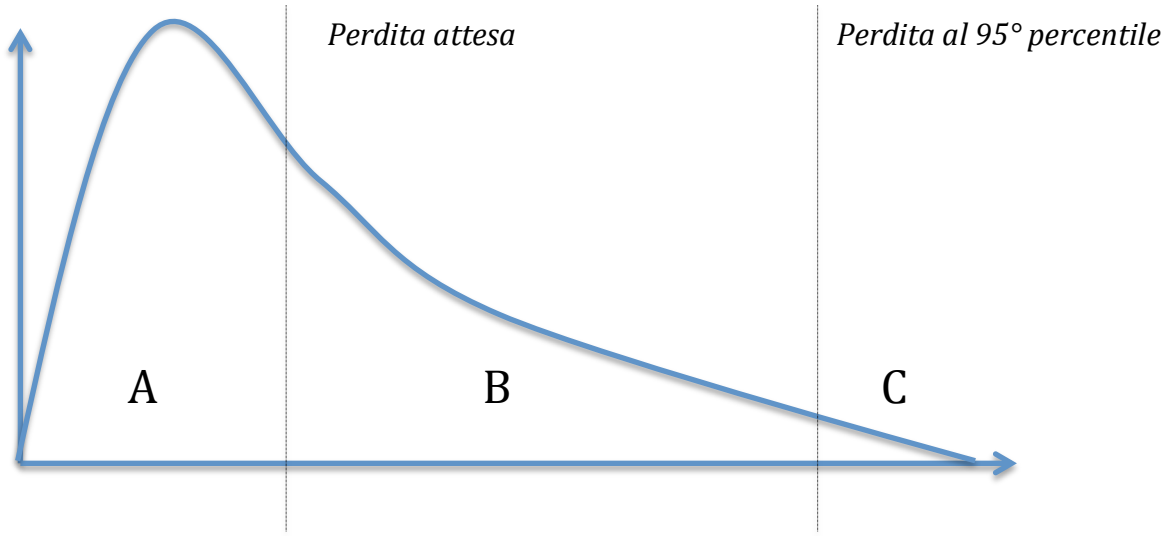
La Perdita attesa EL (Expected Loss) si calcola come:

$$EL = AE * PD * LGD$$

$$EL = AE * PD * (1 - RR)$$

La Perdita inattesa UL (Unexpected Loss) indica il vero fattore di rischio, è una misura di variabilità che può però essere ridotta con un portafoglio ben diversificato.
Diversificare un portafoglio significa ridurre la correlazione tra le perdite inattese dei singoli crediti.

La distribuzione di probabilità delle perdite può essere ricondotta ad una distribuzione con un' asimmetria pronunciata.



La *skewness* è dovuta alla maggior probabilità di perdite di importo limitato e una minore probabilità di eventi che causano perdite molto ingenti.
Si noti come è possibile individuare tre aree distinte nel grafico di distribuzione di probabilità a seconda del valore medio della distribuzione delle perdite e del valore del percentile valutato in base all'intervallo di confidenza $\alpha$:
- A che identifica la Perdita Attesa
- B che identifica la Perdita Inattesa
- C che identifica la Perdita Straordinaria

## *1.3 Perdita Attesa*

### 1.3.1 Adjusted Exposure

Concentrando l'attenzione sulla Perdita attesa, è importante valutare il valore Adjusted Exposure, il valore dell' esposizione in caso di default, poiché non è un valore predefinito e non sempre di facile calcolo.
Nel caso di mutuo è piuttosto facile da calcolare l'AE, il finanziamento è infatti già prescritto, nel caso di fido bancario questo non è più valido, c'è una componente aleatoria, poiché la banca mette a disposizione una somma al cliente, quest'ultimo però può usufruirne quanto e quando vuole.
La frequenza con cui il cliente "sconfina" il massimo del fido caratterizza il già noto rischio di esposizione e ha effetto anche sul rating del cliente.

Ecco che ne deriva una doppia tipologia di esposizione:
- A valore certo: la banca conosce già l'importo
- A valore incerto: la banca può quantificare l'importo solo al manifestarsi dell'insolvenza

Per la stima di AE è necessario conoscere la quota di fido utilizzata (*Drawn Portion*) e la quota non ancora utilizzata (*Updrawn Portion*), in particolare quest'ultima riveste una certa importanza poiché il debitore può cambiare e aumentare l'esposizione in base a quelle che sono le sue condizioni e esigenze.
Al verificarsi dell'insolvenza queste due quote di fido intervengono per misurare effettivamente il valore dell'esposizione ma non è detto che il debitore utilizzi

tutta la *Updrawn Portion*, motivo per cui si introduce una terza variabile (*Usage Given Default*) che rappresenta la percentuale della quota UP che verrà utilizzata dal debitore.
La relazione che lega queste tre grandezze è:

$$AE = DP + UP * UGD$$

### 1.3.2 Loss Given Default

La LGD rappresenta la perdita subita a fronte di un'esposizione quando la controparte diventa insolvente. E' calcolata come il complementare del Recovery Rate RR

$$LGD = 1 - RR$$

Le grandezze RR e LGD sono influenzate da vari fattori:

1) Caratteristiche tecniche del finanziamento:
- Presenza di garanzia reali o finanziarie
- Efficacia delle garanzie
- Grado di seniority

2) Caratteristiche del soggetto finanziato
- Settore produttivo di riferimento
- Regione geografica in cui opera
- Tipologia di contenzioso

3) Fattori interni alla banca
- Procedura ed efficienza del recupero crediti

4) Fattori esterni di tipo "macroeconomico"
- Livello dei tassi di interesse
- Stadio del ciclo economico

La stima di LGD diventa quindi di rilevante importanza, possiamo calcolarla tramite valutazione soggettiva della banca, valutazione dettata da indicazioni di sistema o tramite valutazione proveniente dai mercati finanziari.
In realtà le prime due metodologie non sembrano garantire delle misure corrette, a causa del grado di soggettività nel primo caso e della poca attenzione al soggetto nel secondo; il terzo potrebbe offrire una misura più adatta ma resta la difficoltà di confrontare le attività che sono alla base dell'approccio con quelle presenti nel portafoglio crediti di una banca europea.

Ecco che con quest'equazione è invece possibile stimare il tasso LGD (e di conseguenza RR):

$$LGD = 1 - \frac{\sum_{t=1}^{n} \frac{ER_t - AC_t}{(1+i)^t}}{EAD}$$

dove:
- $ER_t$ indica l'importo recuperato nel periodo t
- $AC_t$ sono i costi amministrativi sostenuti nel periodo t
- $i$ è il tasso di attualizzazione, che può assumere diversi significati, può essere inteso ad esempio come il tasso scritto nel contratto di finanziamento andato in default o come il tasso risk free
- $EAD$ è l'esposizione al momento del default
- $n$ indica il tempo stimato per il recupero

### 1.3.3 Probabilità di Default

La probabilità di default può essere definita una misura diretta della stima del merito creditizio, lo stesso infatti si calcola tramite la probabilità di insolvenza.
Ci sono varie metodologie per il calcolo della PD, si può prevedere la probabilità di default basandosi sul mercato dei capitali, o tramite modelli utilizzando sia aspetti quantitativi/qualitativi che soggettivi.

Monitorare la probabilità di default significa monitorare il rischio, la definizione corretta per la PD fa riferimento anche a casistiche più specifiche e non solo all'insolvenza della controparte.
Secondo la definizione di Standard & Poor's (S&P) il default si ha quando:
- non viene più effettuato il pagamento richiesto di capitali e interessi
- vi è presenza di una procedura concorsuale

• si verifica una ristrutturazione del debito che ne inficia anche il valore totale

Questa definizione, per quanto rilasciata da un ente come S&P non è universale, ciascun intermediario finanziario può infatti avere i propri criteri decisionali e di valutazione del credito purchè rispecchino le linee guida adottate da Basilea II.
Le metodologie più largamente usate fanno riferimento ai rating creditizi che determinate agenzie rilasciano , o ai modelli statistici, come Regressione e algoritmi di Machine Learning.

## *1.4 Perdita Inattesa*

Per il calcolo di perdita inattesa si fa riferimento al modello binomiale che considera solo l'evento default e non default.
• L'evento default si verifica con probabilità PD e genera una perdita pari a LGD
• L'evento non default è quindi l'esatto complemento e si verifica con probabilità (1-PD) e genera una perdita nulla.

Ecco che definendo la media e la volatilità di questi due eventi vediamo come la perdita attesa altro non è che la media di questo modello binomiale, di contro la volatilità identificherà la perdita inattesa.

$$\mu = Perdita\ Attesa = PD * LGD + (1 - PD) * 0 = PD * LGD$$
$$\sigma = Perdita\ Inattesa = LGD * \sqrt{PD * (1 - PD)}$$

Se si tiene conto anche della volatilità del tasso di perdita e non si considera correlazione tra PD e LGD ecco che la perdita inattesa diventa:

$$UL = \sqrt{PD * (1 - PD) * LGD^2 + PD * \sigma_{LGD}^2}$$

La distinzione tra i due tipi di perdita è di notevole rilevanza dal punto di vista economico-contabile poiché mentre per quella attesa si considera un accantonamento a riserva registrato in conto economico, per quella inattesa si stima una copertura da patrimonio della banca, di conseguenza il patrimonio degli azionisti verrà intaccato.

# 2. MODELLI DI REGRESSIONE

## *2.1 BIAS – Valutazione dell'accuratezza del modello*

Il concetto di accuratezza è ciò che fa la differenza nella scelta dell'utilizzo di un modello, come vedremo infatti, una volta individuato quello che meglio riesce a rappresentare il nostro dataset, lavoreremo affinchè restituisca il miglior risultato possibile, e lo faremo misurando l'*accuracy*.

Una prima misura che permette la valutazione del modello, andando a vedere quanto le previsioni effettuate si discostino dalle osservazioni effettive è il *mean squared error* (MSE), definito come:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

dove $\hat{f}(x_i)$ rappresenta la predizione che il modello restituisce per l'ossevazione *i.*
Quanto più le previsioni sono vicine alle osservazioni quanto più il valore MSE sarà piccolo, al contrario a valori maggiori di MSE corrisponderà un errore di predizione più grande.
Il valore di MSE è calcolato modellando i dati del training data, sarebbe più corretto dunque chiamarlo training MSE, ma non siamo interessati a vedere quanto il modello performi bene sui training data. Dobbiamo misurare l'accuratezza delle previsioni, quindi sarà necessario utilizzare i testing data.
Da un punto di vista matematico, volendo costruire il nostro modello statistico sui nostri training data :

$$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$$

otteniamo le rispettive previsioni

$$\hat{f}(x_i)\ con\ i = 1, \dots, n$$

se quindi queste previsioni hanno valore simile alle osservazioni $y_i$ allora il valore MSE sarà piccolo, ma non siamo interessati a valutare se $\hat{f}(x_i) \approx y_i$, vogliamo invece verificare se $\hat{f}(x_0) \approx y_0$ dove $(x_0, y_0)$ rappresentano le previsioni risultanti dall' applicazione del modello statistico sui testing data.
Quindi ci interessa ottenere il più basso valore possibile per il test MSE, e volendo ricavare una misura d'insieme calcoliamo:

$$Avg(y_0 - \hat{f}(x_0))^2$$

ovvero la media dell'errore quadratico delle previsioni.

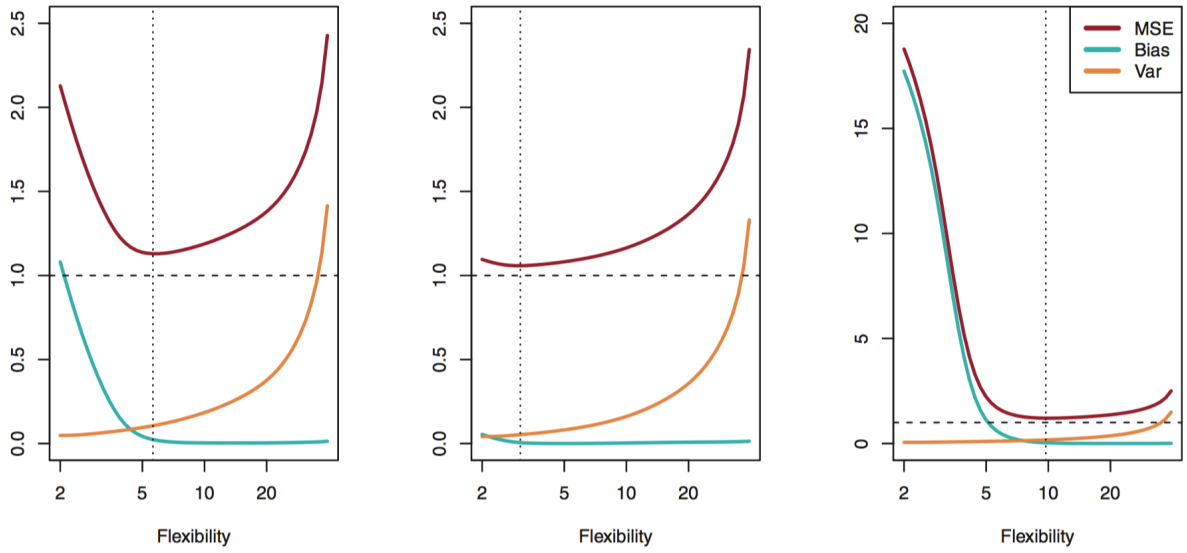
Il test MSE è composto da tre quantità fondamentali, che sono:
- La varianza della previsione dei testing data $\hat{f}(x_0)$
- La distorsione bias $Bias(\hat{f}(x_0))$
- La varianza dell'errore $\epsilon$

$$E(y_0 - \hat{f}(x_0))^2 = Var\left(\hat{f}(x_0)\right) + [Bias\left(\hat{f}(x_0)\right)]^2 + Var(\epsilon)$$

Il termine a destra dell'equazione si riferisce al risultato medio del test MSE dopo aver adottato il modello di previsione su più training data.
L'obiettivo è quindi quello di minimizzare l'errore del test ottenendo valori di varianza e bias più bassi possibile, ma cosa si intende precisamente per varianza e bias?
Con varianza intendiamo di quanto la previsione $\hat{f}$ si discosta se utilizziamo un training dataset differente, questo perché dal momento che i training data sono utilizzati proprio per creare il modello d'apprendimento, diversi training data costituiscono diversi testing data.
Mentre con il termine Bias si intende un errore che si genera qualora cerchiamo di modellare un problema particolarmente complicato utilizzando un modello più semplice.

Si possono quindi verificare casi dove il nostro modello realizza errori bias alti ma con bassa varianza, in generale più il metodo predittivo è flessibile, maggiore sarà la varianza e minore il bias.

Queste due grandezze sono strettamente correlate tra di loro, si parla infatti di *variance-bias trade-off,* e per evidenziare l'effetto che ognuna ha sul valore finale MSE introduciamo il seguente grafico:




I tre grafici in esame corrispondono a tre dataset iniziali diversi, e vediamo come il risultato finale MSE cambi a seconda dei valori di varianza e bias che il modello genera.

La linea blu rappresenta il Bias quadratico $[Bias\left(\hat{f}(x_0)\right)]^2$, e vediamo che, all'aumentare della flessibilità, diminuisce per tutti i dataset; comportamento opposto invece per la linea arancio che rappresenta la varianza $Var\left(\hat{f}(x_0)\right)$ generata dal modello, essa tende ad aumentare con l'aumento della flessibilità, per tutti i casi.
Ma il risultato finale (che è dato dalla somma di queste due componenti) non è lo stesso per le tre casistiche, sia perché c'è una componente di errore generico da considerare nella realizzazione del modello ma soprattutto perché dipende dalle condizioni di linearità del dataset messo a disposizione, a seconda quindi se può essere modellato da una regressione lineare (come nel secondo grafico) o se il modello di regressione lineare genera scarsi risultati (è il caso del terzo grafico)

## 2.2 Odds e Log(Odds)

Prima di introdurre il modello Logit introduciamo il concetto di Odds e Log(Odds) che sono importanti sia per il modello Logit quanto per algoritmi di Boosting.
Le Odds di un evento sono rappresentate dal rapporto del numero di realizzazioni di un evento $E_i$ che ci interessa calcolare sul numero di realizzazioni dell' evento complementare (-$E_i$), quindi:

$$Odds = \frac{N(E_i)}{N(-E_i)}$$

da non confondere con la probabilità che si verifichi l'evento $E_i$, che chiaramente sarà:

$$P(E_i) = \frac{N(E_i)}{N(E_i) + N(-E_i)}$$

la relazione che lega Odds e Probabilità invece è descritta da questa semplice equazione:

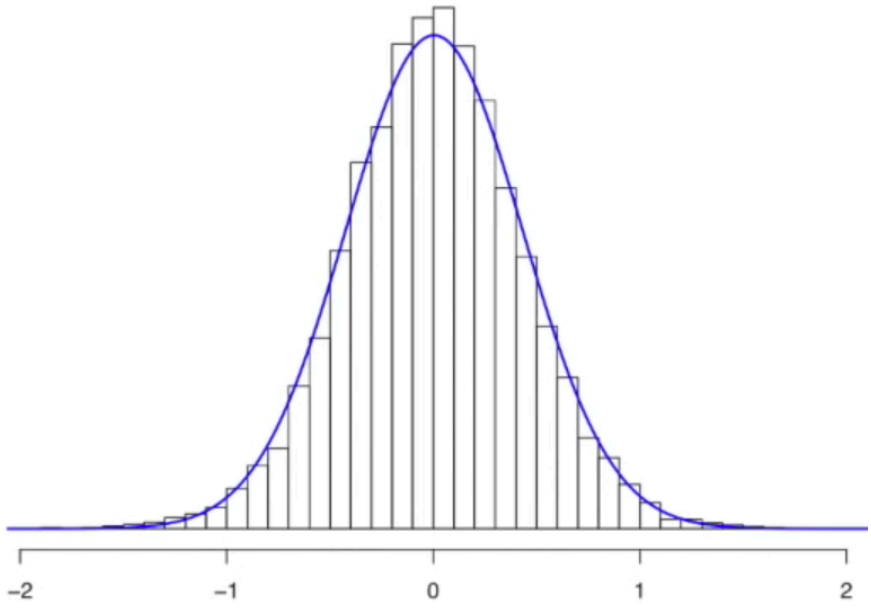
$$Odds = \frac{P(E_i)}{1 - P(E_i)}$$

Introduciamo ora il concetto di log(odds).
Notiamo che avere le odds contro restituisca sempre un valore $0 < x < 1$, con $x$ tendente a zero quanto più l'evento sia sfavorevole, avere invece le odds a favore restituisce sempre un valore positivo, questo perché $N(E_i) > N(-E_i)$ .
Ma si noti come la distribuzione delle odds non risulta simmetrica, in quanto le odds sfavorevoli saranno tutte concentrate nell' intervallo (0,1) mentre le odds favorevoli a seconda di quanto siano favorevole l'evento possono teoricamente avere un valore tendente ad infinito.
Motivo per cui si introduce la grandezze log(odds), in modo da "nomalizzare" i valori delle odds e avere come valore centrato il valore nullo.
Ecco quindi che

$$\log(Odds) = \log\left(\frac{N(E_i)}{N(-E_i)}\right) = \log\left(\frac{P(E_i)}{1 - P(E_i)}\right)$$

Proprio la grandezza $\log\left(\frac{p}{1-p}\right)$ sarà alla base della regressione logistica.

Qual è quindi il vantaggio di eseguire il log delle Odds? Se rappresentiamo i risultati dei log(odds) tramite istogramma vediamo come sia possibile modellarli con una distribuzione Normale, e il vantaggio immediato è quello di poter usare il log(odds) per risolvere certi problemi statistici con variabili dicotomiche, in situazioni del tipo win/lose, si/no, o anche BuonCredito/CattivoCredito.



## 2.3Modello Logistico (o Logit)

Tra i modelli più utilizzati per la valutazione del rischio di credito ci sono quelli di tipo statistico, conosciuti come modelli di scoring.
Questi utilizzano degli indicatori economici (relativi alle performance aziendali) cui sono assegnati diverse ponderazioni a seconda della loro correlazione con l'evento default, l'obiettivo è ottenere un valore numerico chiamato score che sintetizza la probabilità di insolvenza.
Il Modello Logistico segue questo approccio e viene applicato quando la variabile di riferimento è di tipo dicotomico.

La sua costruzione prevede quattro fasi principali:
1) Selezione del campione
2) Selezione delle variabili esplicative
3) Stima del modello
4) Valutazione del modello

Per costruire il modello di regressione logistica possiamo partire analizzando il modello di regressione lineare semplice e proviamo a modellarlo utilizzando come output la probabilità dell'evento X che vogliamo considerare e inserendo solo le variabili 1,0 ad indicare se si verifica o meno l'evento default:

$$P(X) = \beta_0 + \beta_1 X$$

Il risultato che si ottiene è sempre una retta di regressione ma senza senso matematico, poiché andando a tracciare la retta osserviamo che è possibile registrare anche valori di probabilità negativi, ma dal momento che $P(X)$ deve necessariamente essere $\geq 0$ , concludiamo che dobbiamo apportare per forza delle modifiche al modello lineare.
Per iniziare infatti si realizza un modello che restituisca output (in questo caso proprio le probabilità che si verifichi l'evento default) compreso nell' intervallo [0,1]
Utilizziamo quindi la regressione logistica:

$$P(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

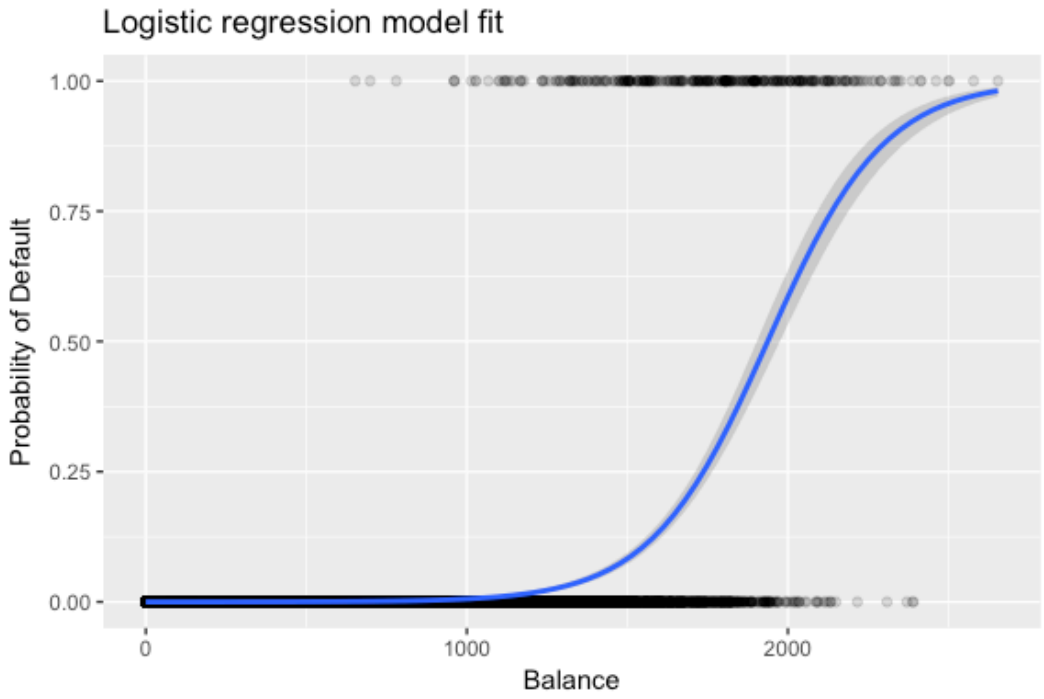
da qui si ricava:

$$\frac{P(X)}{1-P(X)} = e^{\beta_0+\beta_1 X}$$

il termine a sinistra dell'equazione indica proprio le odds, che possono avere valore da 0 a ∞, rispettivamente ad indicare basse e alte probabilità di default.
Ma come abbiamo visto prima, possiamo passare dalle odds a log(odds) ottenendo:

$$\log\left(\frac{P(X)}{1-P(X)}\right) = \beta_0 + \beta_1 X$$

La realizzazione di questo modello parte dal metodo di massima verosimiglianza e il risultato è una curva ad S caratteristica di tutte le regressioni logistiche.
Notiamo come i valori di probabilità siano giustamente positivi e compresi nell'intervallo [0,1] e come per ogni variabile si possa associare un valore di probabilità sulla curva.



Nell'equazione del modello di regressione logistica compaiono i coefficienti $\beta_0$ e $\beta_1$, essi non sono valori noti e vanno stimati in base ai dati a disposizione.
Nel modello di regressione lineare i coefficienti sono stimati con il metodo dei minimi quadrati, ma come visto, per problemi di classificazione e con variabili dicotomiche, quindi con il modello di regressione logistica, si stimano con il metodo della massima verosimiglianza.
In particolare, conoscendo le probabilità di default (in base alla variabili di input X) si modellano i coefficienti $\beta_0$ e $\beta_1$ in modo da ottenere $P(X) = 1$ per gli eventi di default e un valore prossimo allo zero per le osservazioni per cui non si è verificato default.
Questa stima è calcolata con la funzione di verosimiglianza:

$$v(\beta_0, \beta_1) = \prod_{i:y_i=1} P(X_i) \prod_{i':y_{i'}=0} (1 - P(X_{i'}))$$

I valori $\beta_0$ e $\beta_1$ stimati sono quelli che massimizzano questa funzione. Il metodo di massima verosimiglianza si sposa bene con il modello di regressione logistica, dal momento che si utilizza per stimare valori utilizzati in modelli non lineari.

Per la previsione della probabilità finale si utilizza l'equazione vista per il modello di regressione logistica:

$$P(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

Se abbiamo più predittori si parla di regressione logistica multipla, il procedimento è analogo a quello della regressione logistica semplice, cambia chiaramente il modello, quindi il calcolo della probabilità e dei log(odds), che diventano rispettivamente:

$$P(X) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_P X_P}}{1 + e^{\beta_0+\beta_1 X_1+\cdots+\beta_P X_P}}$$

$$\log\left(\frac{P(X)}{1 - P(X)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_P X_P$$

## 2.4 Valutazione del modello

Una volta ottenuti i risultati con il modello logit, occorre che questi risultati siano quanto più vicini possibile ai valori osservati, in modo da misurare l'accuratezza del modello.
La valutazione si può effettuare in vari modi:

1) Confronto tra il modello stimato e il modello "effettivo", che contiene tanti parametri quante osservazioni e permette il miglior adattamento.
Il confronto è effettuato con la statistica Deviance D, in particolare si mettono a confronto le stime ottenute tramite massima verosimiglianza dei parametri, ed è definita come:

$$D = -2\log\left(\frac{v_s}{v_{eff}}\right)$$

questa grandezza risulterà sempre positiva poiché:

$$v_s < v_{ee} \Longrightarrow \frac{v_s}{v_{eff}} < 1$$

L'obiettivo sarebbe riuscire ad azzerare la Deviance D, questo significherebbe che:

$$v_s \approx v_{eff}$$

perchè quanto più la stima calcolata si avvicina al modello effettivo quanto più la Deviance assume valori minori a indicare un buon adattamento.

2) La statistica Pseudo - $R^2$ che costituisce un estensione del coefficiente $R^2$, è utilizzata appunto per il modello logit e misura la variabilità dei dati spiegata dal modello in un intervallo (0,1), dove 1 indica il miglior adattamento possibile.
Indicando con $v_s$ la verosimiglianza del modello stimato e con $v_0$ quella del modello con singola intercetta, la statistica Pseudo - $R^2$ è definita come:

$$\text{Pseudo} - R^2 = 1 - \frac{log(v_s)}{log(v_0)}$$

3) La statistica di Wald che valuta la significatività del parametro $\beta_i$
Le ipotesi da verificare sono:

H0: $\beta_i = 0$
H1: $\beta_i \neq 0$

La statistica utilizzata è:

$$W = \frac{\beta_i}{\sigma(\widehat{\beta_J})}$$

dove $\sigma(\widehat{\beta_J})$ indica l'errore standard della stima del parametro.
Definendo un livello di significatività $\alpha$, rifiuto l'ipotesi nulla se:

$$|W| > z_{\alpha/2}$$

quindi rifiuto l'ipotesi secondo cui non ci siano variabili significative per il modello.

## 2.5 Regressione Ridge e LASSO

La Ridge Regression è un metodo utilizzato per selezionare automaticamente le variabili significative per il modello che stiamo creando. E' un metodo molto simile a quello dei minimi quadrati ma ci sarà in più un termine di regolarizzazione che penalizzerà la complessità del modello.
Insieme al LASSO è un modello di regressione che viene ripreso da tecniche di Boosting di Machine Learning, dove vedremo sarà presente un termine di regolarizzazione, di *shrinkage,* il cui significato matematico può essere appunto spiegato da Ridge e LASSO regression.

La differenza principale tra i 2 modelli di regressione sta nel termine di regolarizzazione che attribuisce penalità al modello:

- Ridge Regression: $\lambda \sum_{j=1}^{p} \beta_j^2$
- LASSO Regression: $\lambda \sum_{j=1}^{p} |\beta_j|$

### *2.5.1 Regressione Ridge*

La Ridge Regression è indicata per i problemi di multicollinearità, che è un fenomeno in cui la variabile predittiva in un modello di regressione multipla può essere prevista linearmente dalle altre con un sostanziale grado di accuratezza.
Questo può creare stime non precise in quanto cambiando anche di poco il dataset inziale si rischia di avere delle misure predittive completamente diverse e anche poco efficaci.

Richiamando il metodo dei minimi quadrati abbiamo:

$$RSS = \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2$$

La Regressione Ridge non è molto diversa dal metodo dei minimi quadrati, tranne per il fatto che i coefficienti sono stimati minimizzando una quantità diversa, in effetti la particolarità di questi modelli di regressione è l'aggiunta di un parametro di regolarizzazione, otteniamo quindi:

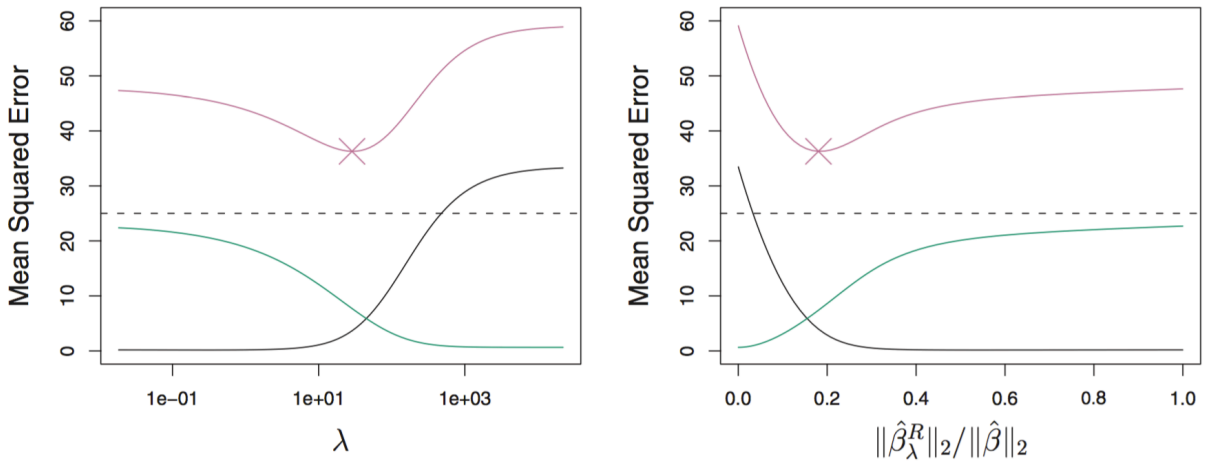
$$\sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \beta_j^2 = RSS + \lambda \sum_{j=1}^{p} \beta_j^2$$

Per il la stima dei coefficienti quindi si utilizza il metodo dei minimi quadrati in aggiunta al termine $\lambda \sum_{j=1}^{p} \beta_j^2$ che è piccolo quando $\beta_1, \beta_2, \dots, \beta_n$ sono prossimi allo zero, il parametro $\lambda$ serve proprio a controllare l'effetto di questi due termini nella stima dei coefficienti di regressione.
Un valore $\lambda = 0$ non avrà effetto sul risultato finale, l'equazione infatti viene ricondotta a quella dei minimi quadrati, nel caso $\lambda$ tenda a infinito i coefficienti di

regressione stimati tenderanno a zero poiché si darà molto peso alla regolarizzazione/penalità del modello.

Si noti come il termine $\lambda$ è applicato solo a $\beta_1, \beta_2, \dots, \beta_n$ ma non all' intercetta $\beta_0$, questo perché vogliamo regolarizzare la correlazione tra ogni variabile e la risposta ma non l' intercetta che si ottiene quando $x_{i1}, x_{i2}, \dots, x_{ip} = 0$.

Il modello di Ridge Regression ha dei vantaggi rispetto a quello dei minimi quadrati, soprattutto per quanto riguarda il *bias-variance trade-off.*
Quando $\lambda$ aumenta la flessibilità del modello diminuisce, questo comporta una riduzione della varianza ma un aumento del bias. Vediamo un esempio:




Il grafico a sinistra mostra il bias(nero) , la varianza(verde) e il test MSE(viola) in relazione a $\lambda$. Il grafico a destra mostra invece questi tre termini in relazione a $\left\|\widehat{\beta_\lambda^R}\right\|_2/\left\|\hat{\beta}\right\|_2$, dove $\hat{\beta}$ rappresenta il vettore dei coefficienti stimati con metodo dei minimi quadrati, e $\|\beta\|_2$ indica la cosiddetta $l_2\, norm$ che è definita come $\|\beta\|_2 = \sqrt{\sum_{j=1}^{p} \beta_j^2}$, quest'ultima diminuirà sempre al diminuire del parametro $\lambda$ e misura la "distanza" di $\beta$ da zero.
Il range di $\left\|\widehat{\beta_\lambda^R}\right\|_2/\left\|\hat{\beta}\right\|_2$ varia da 1( quando $\lambda = 0$ e quindi la regressione ridge stima i coeficienti allo stesso modo dei minimi quadrati) fino a 0( quando $\lambda = 1$ e quindi la regressione ridge tende a stimare i coefficienti con valori sempre più vicini allo zero).

In generale, quando c'è una relazione lineare tra i predittori e la variabile risposta, il modello dei minimi quadrati comporta poco bias ma alta varianza.

Questo si traduce nel fatto che una piccola variazione nel training data può generare un cambiamento notevole nei coefficienti stimati, di contro la ridge regressione lavora bene nelle situazioni dove il modello dei minimi quadrati genera ampia varianza nelle stime.

### *2.5.1 Regressione LASSO*

Lo svantaggio della regressione Ridge è il fatto di considerare tutte le variabili per la predizione nel modello finale. Il termine di regolarizzazione $\lambda \sum_{j=1}^{p} \beta_j^2$ tende ad assegnare ai coefficienti valori vicini allo zero, ma non perfettamente zero, a meno che $\lambda = 0$.
Questo non crea problemi per l' accuratezza della predizione quanto per l'interpretazione delle varabili, soprattutto quando il numero delle variabili diventa alto.
La regressione LASSO(che sta per *least absolute shrinkage and selection operator*) è un'alternativa alla regressione ridge utilizzata proprio per superare questo problema.
La differenza sta nel termine di regolarizzazione, che nel caso di LASSO regression è:

$$\lambda \sum_{j=1}^{p} |\beta_j|$$

e l'equazione del modello diventa:

$$\sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} |\beta_j| = RSS + \lambda \sum_{j=1}^{p} |\beta_j|$$

Anche nel caso di LASSO regression il parametro di regolarizzazione tende a stimare i valori dei coefficienti verso lo zero ma, a differenza della regressione ridge, la "penalità" $\lambda \sum_{j=1}^{p} |\beta_j|$ costringe uno o più coefficienti ad essere esattamente zero per certi valori di $\lambda$.

I modelli di regressione Ridge e LASSO possono essere rispettivamente riscritti come:

$$argmin\{\sum_{i=1}^{n}(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij})^2\} \quad s.t. \quad \sum_{j=1}^{p}|\beta_j| \leq s$$

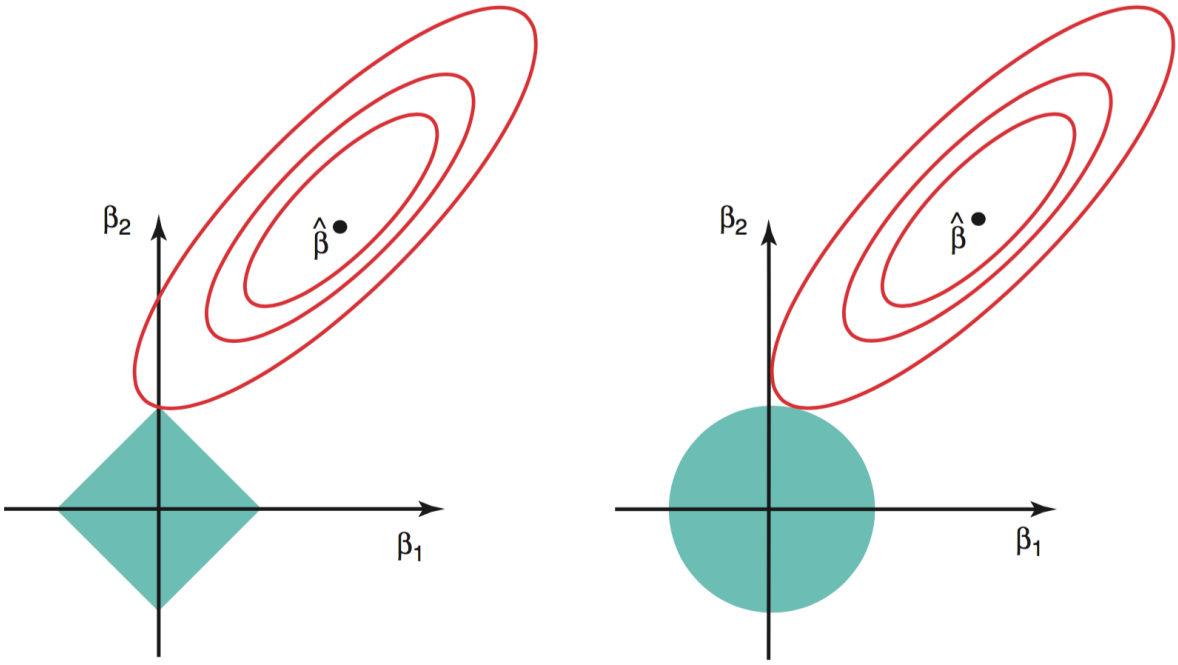
$$argmin\{\sum_{i=1}^{n}(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij})^2\} \quad s.t. \quad \sum_{j=1}^{p}\beta_j^2 \leq s$$

Per ogni valore di $\lambda$ esisterà $s$ tale che le equazioni dei modelli di Ridge e LASSO con le rispettive riformulazioni appena proposte daranno gli stessi coefficienti stimati.

Queste riformulazioni possono essere utili per evidenziare il motivo per cui la regressione LASSO a differenza di quella Ridge possa stimare dei coefficienti in modo che siano esattamente pari a zero, e questo si osserva direttamente dai due vincoli:

$$\sum_{j=1}^{p}|\beta_j| \leq s \quad , \quad \sum_{j=1}^{p}\beta_j^2 \leq s$$

Rappresentando la situazione nella seguente figura, notiamo come in entrambe compare la stessa soluzione, quindi la stessa stima $\hat{\beta}$ ottenuta dal metodo dei minimi quadrati:



Le regioni in verde sono le aree delimitate dai rispettivi vincoli e notiamo come hanno forme diverse, questo perché nel caso di regressione LASSO (sinistra) il vincolo sarà:

$$|\beta_1| + |\beta_2| \leq s$$

nel caso di regressione Ridge (destra) il vincolo sarà:

$$\beta_1^2 + \beta_2^2 \leq s$$

Nel caso in cui $s$ fosse sufficientemente grande, $\hat{\beta}$ cadrebbe all'interno di entrambe le aree per cui entrambe le regressioni avrebbero lo stesso valore del metodo dei minimi quadrati, ma non è questo il caso.
Le ellissi centrate in $\hat{\beta}$ rappresentano regioni dove RSS è costante, tutti i punti appartenenti ad una stessa ellisse condividono quindi lo stesso valore RSS.

Le equazioni per regressione Ridge e LASSO:

$$argmin\{\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2\} \quad s.t. \sum_{j=1}^{p}|\beta_j| \leq s$$

$$argmin\{\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2\} \quad s.t. \sum_{j=1}^{p}\beta_j^2 \leq s$$

indicano che i coefficienti stimati si ottengono dal punto di intersezione tra la regioni delimitate dai rispettivi vincoli e dalle ellissi. Dal momento che l'area individuata dalla regressione Ridge è una circonferenza, il primo punto individuato dall' intersezione non sarà mai sull'asse e questo si traduce in un coefficiente stimato sicuramente diverso da zero che al più può tendere e avvicinarsi a zero.
Al contrario la regressione LASSO individua un'area che genera un punto di intersezione con l'ellisse proprio sull'asse, è il motivo per cui riesce ad individuare coefficienti esattamente uguali a zero.

## 2.6 Confronto Ridge e LASSO

Per comprendere meglio il ragionamento dietro regressione Ridge e LASSO consideriamo una matrice X con valori pari a 1 solo lungo la diagonale principale, per il resto tutti valori nulli.
Supponiamo inoltre che di voler modellare una regressione senza intercetta, questo si traduce nell' assenza del coefficiente $\beta_0$.
Applichiamo il metodo dei minimi quadrati per stimare i parametri $\beta_1, \beta_2, \dots, \beta_n$ minimizzando:

$$\sum_{j=1}^{p} (y_j - B_j)^2$$

In questo caso la soluzione che minimizza il problema è data da:

$$\widehat{\beta_J} = y_j$$

Applicando i modelli di regressione Ridge e LASSO ottengo invece rispettivamente:

$$\sum_{j=1}^{p} (y_j - B_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

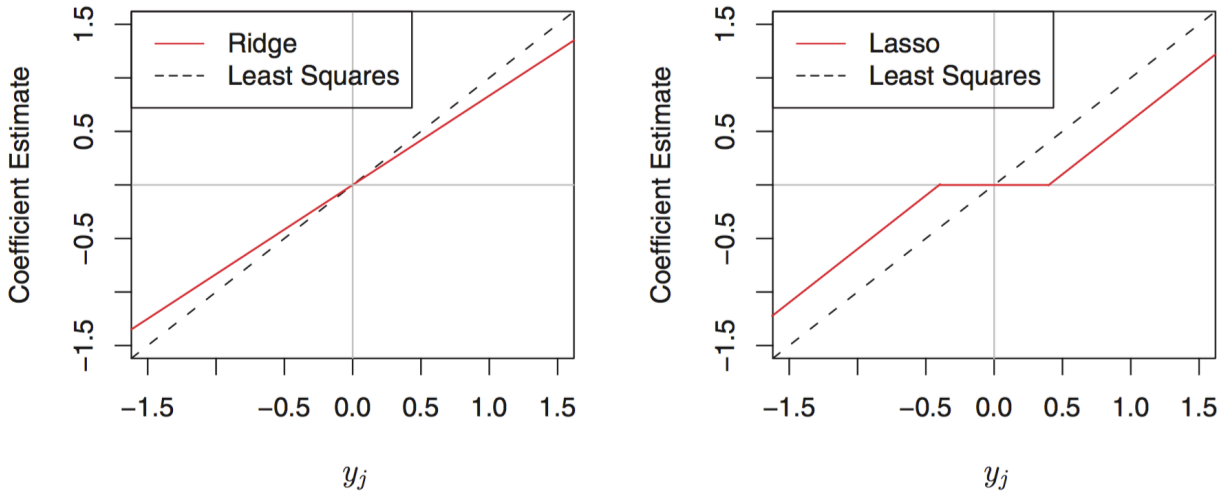
$$\sum_{j=1}^{p} (y_j - B_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Risolvendo, i coefficienti stiamati dai due modelli saranno rispettivamente:

$$\widehat{\beta_J^R} = \frac{y_j}{(1+\lambda)}$$

$$\widehat{\beta_J^L} = \begin{cases} y_j - \frac{\lambda}{2} \;\; se \;\; y_j > \frac{\lambda}{2} \\ y_j + \frac{\lambda}{2} \;\; se \;\; y_j < -\frac{\lambda}{2} \\ 0 \;\; se \;\; |y_i| \leq \lambda/2 \end{cases}$$

Rappresentando la situazione su un grafico otteniamo:




Notiamo come i due modelli di regressione performano in modo diverso ma soprattutto applicano due diverse regolarizzazioni al modello.
Nella regressione Ridge ogni coefficiente stimato è regolarizzato nella stessa proporzione, quello che si ottiene infatti è una linea retta con una pendenza diversa rispetto al risultato stimato ottenuto con il metodo dei minimi quadrati.
Nel caso della regressione LASSO invece, la regolarizzazione è costante per i coefficienti stimati vicino allo zero, ma è esattamente uguale a zero per quei valori $y_i$ che in modulo sono minori di $\lambda/2$.

Per quanto i modelli effettivi siano ben più complicati, questo esempio semplificativo rende bene la differenza tra i due modelli di regressione:
- Il modello Ridge tende a regolarizzare i coefficienti nella stessa proporzione
- Il modello LASSO regolarizza i coefficienti di un valore costante quando si avvicinano allo zero e per un certo valore di $\lambda$ stima tutti i coefficienti uguali a zero.

## 2.7 Matrici di Classificazione e Curve ROC

Definito il modello e realizzate le previsioni si passa all'analisi delle performance.
A seconda del modello statistico utilizzato possono esserci diverse tipologie di valutazioni indicate.

Considerando i problemi di classificazione sul rischio di credito, valuteremo in particolare le matrici di classificazione e le curve ROC, queste due analisi sono strettamente collegate tra di loro, e soprattutto le prime saranno alla base del calcolo dell'accuracy dei vari modelli che costruiremo su R utilizzando algoritmi di Machine Learning.

### *2.7.1 Matrice di classificazione*

In un problema di classificazione cerchiamo di costruire un modello che restituisca delle previsioni accurate. Quando si parla di previsioni accurate ci si riferisce a un concetto molto semplice, si verifica in base alle osservazioni del testing data se le previsioni effettuate coincidono con i valori che abbiamo nel dataset.
Quindi, considerando i risultati come positivi e negativi (ma il problema di classificazione potrebbe riguardare qualsiasi variabili dicotomica, come SI/NO, BuonCredito/CattivoCredito ecc) vogliamo che le previsioni positive siano quanto più vicine in numero alle osservazioni positive del dataset, stesso ragionamento per le previsioni negative.
Il modello di previsione, se ben strutturato, può dare un'accuratezza alta, il fatto di non ottenere il 100% di accuratezza indica che non tutte le previsioni sono effettivamente corrette, un'osservazione positiva può risultare negativa dal modello di previsione e viceversa.
Per riportare quindi l'esempio sui rischi di credito, un'azienda sana o un buon credito può dare una previsione sbagliata, indicando quindi un'azienda anomala o un cattivo credito, l'errore nella previsione non è il vero problema, occorre però modellare i dati al fine di avere delle previsioni quanto più accurate, in modo che questi errori di misclassificazione si verifichino il meno possibile.
Ecco che dal modello di previsione del problema di classificazione possiamo ottenere quattro possibili output: Veri Positivi, Veri Negativi, Falsi Positivi, Falsi Negativi.

Possiamo rappresentare i risultati nella cosiddetta matrice di classificazione:

| | | PREDIZIONE | |
|---|---|---|---|
| | | *Positivo* | *Negativo* |
| OSSERVAZIONE | *Positivo* | **Veri Positivi (A)** | **Falsi Negativi (C)** |
| | *Negativo* | **Falsi Positivi (B)** | **Veri Negativi (D)** |

Da questa matrice si possono individuare diverse misure per stabilire l'accuratezza del modello, e nella sua semplicità riesce a dare dei risultati immediati per verificare se l'accuratezza delle previsioni ci soddisfa o meno.
Si pensi all'importanza della matrice di classificazione in un modello ad albero decisionale, abbiamo la variabile da prevedere che può assumere due valori distinti, e possiamo misurarne l'accuratezza riportando i dati ottenuti dalla struttura ad albero (precisamente dai nodi foglia come vedremo meglio) in una matrice 2x2.

Le grandezze più note che possiamo ottenere analizzando la matrice sono:

• Frazione dei Veri Positivi, noto come TPR(*True Positive Rate*) o meglio Sensitività.
La sensitività è la proporzione delle unità che hanno osservazione positiva, e rappresenta l'asse y della Curva ROC

| | |
|---|---|
| **Veri Positivi (A)** | **Falsi Negativi (C)** |
| **Falsi Positivi (B)** | **Veri Negativi (D)** |

$$Sensitività = \frac{A}{A + C}$$

• Frazione dei veri negativi, noto come TNR (*True Negative Rate*) o Specificità.
La Specificità è la proporzione delle unità con un'osservazione nota negativa. Il complemento a questa grandezza rappresenta l'asse x della Curva ROC

| | |
|---|---|
| **Veri Positivi (A)** | **Falsi Negativi (C)** |
| **Falsi Positivi (B)** | **Veri Negativi (D)** |

$$Specificità = \frac{D}{D + B}$$

Come abbiamo detto la matrice di classificazione può essere riscritta utilizzando le variabili di nostro interesse, e un'osservazione interessante va fatta per i Falsi Positivi e i Falsi Negativi, che possono essere visti rispettivamente come l'Errore di I° tipo e l'Errore di II° tipo, possiamo rivedere la nostra matrice in questo modo:

| | | PREDIZIONE | |
|---|---|---|---|
| | | *Società Sana* | *Società Anomala* |
| OSSERVAZIONE | *Società Sana* | **OK** | **Errore di II° tipo** |
| | *Società Anomala* | **Errore di I° tipo** | **OK** |

Altri valori interessanti che si possono ricavare dalla matrice di classificazione sono:

• Predizione del Positivo, meglio noto come Precisione.
La Precisione è la proporzione dei dati con una previsione positiva, distinguendo quindi tra i Veri Positivi e i Falsi Positivi

| | |
|---|---|
| **Veri Positivi (A)** | **Falsi Negativi (C)** |
| **Falsi Positivi (B)** | **Veri Negativi (D)** |

$$Precisione = \frac{A}{A + B}$$

Introduciamo ora la Prevalenza, essa indica la proporzione di una popolazione che ha una specifica caratteristica nel momento in cui la si misura. Quando la prevalenza è nota, quindi in fase di pre-test si conosce la proporzione della condizione positiva, la Precisione può essere corretta utilizzando il teorema di Bayes, ottenendo quindi:

$$Precisione\ Corretta = \frac{Sensitività * prevalenza}{sensitività * prevalenza + (1 - specificità) * (1 - prevalenza)}$$

Lo stesso ragionamento può essere fatto calcolando la Predizione del Negativo e anche in questo caso se la Prevalenza dell'osservazione negativa è nota si può correggere con il teorema di Bayes

Introduciamo ora le ultime grandezze ricavabili dalla matrice utilizzando tutte le unità, possiamo calcolare la:

• Accuratezza delle classificazioni corrette, andiamo quindi a misurare quanto le previsioni del modello sono state accurate

| | |
|---|---|
| **Veri Positivi (A)** | **Falsi Negativi (C)** |
| **Falsi Positivi (B)** | **Veri Negativi (D)** |

$$Accuratezza = \frac{A + D}{A + B + C + D}$$

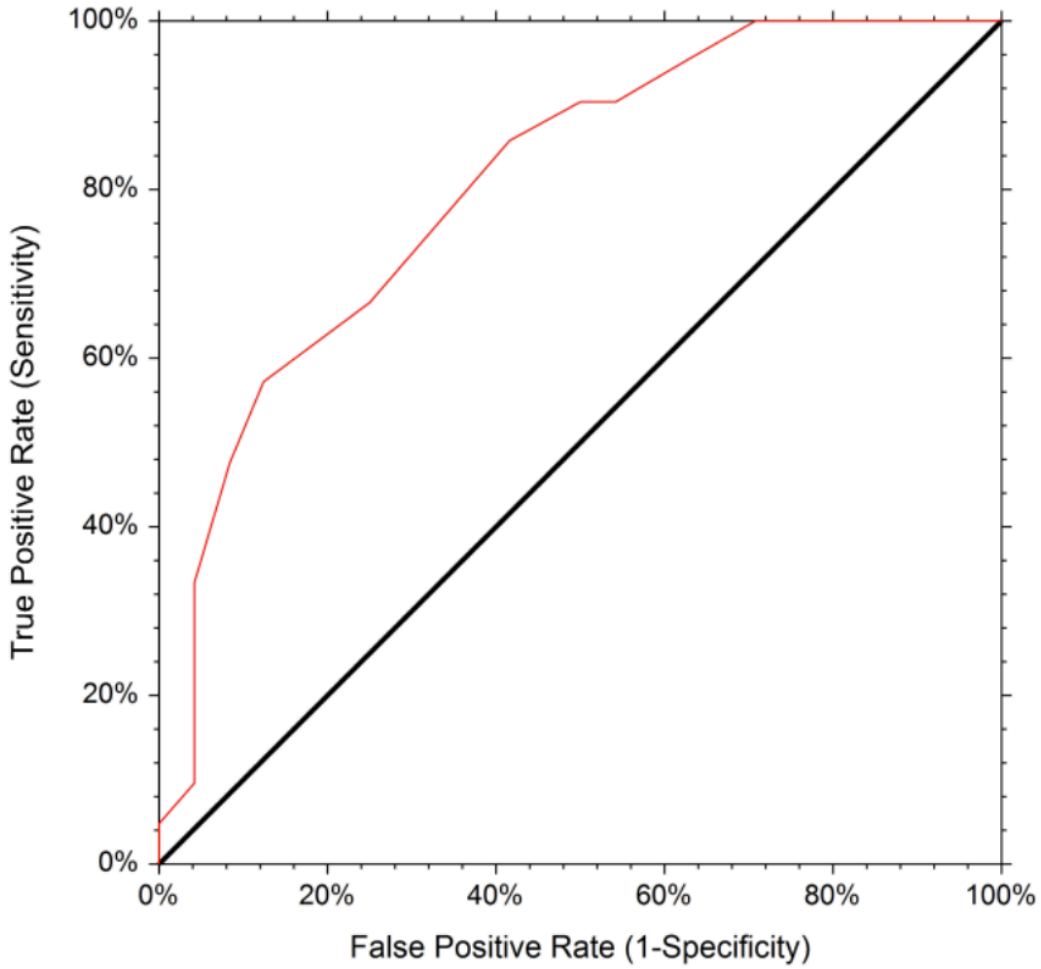
• Di contro si può misurare l'errore generato dalla matrice per i dati non correttamente classificati

| | |
|---|---|
| **Veri Positivi (A)** | **Falsi Negativi (C)** |
| **Falsi Positivi (B)** | **Veri Negativi (D)** |

$$Errore\ di\ Classificazione = \frac{B + C}{A + B + C + D}$$

### *2.7.2 Curva ROC*

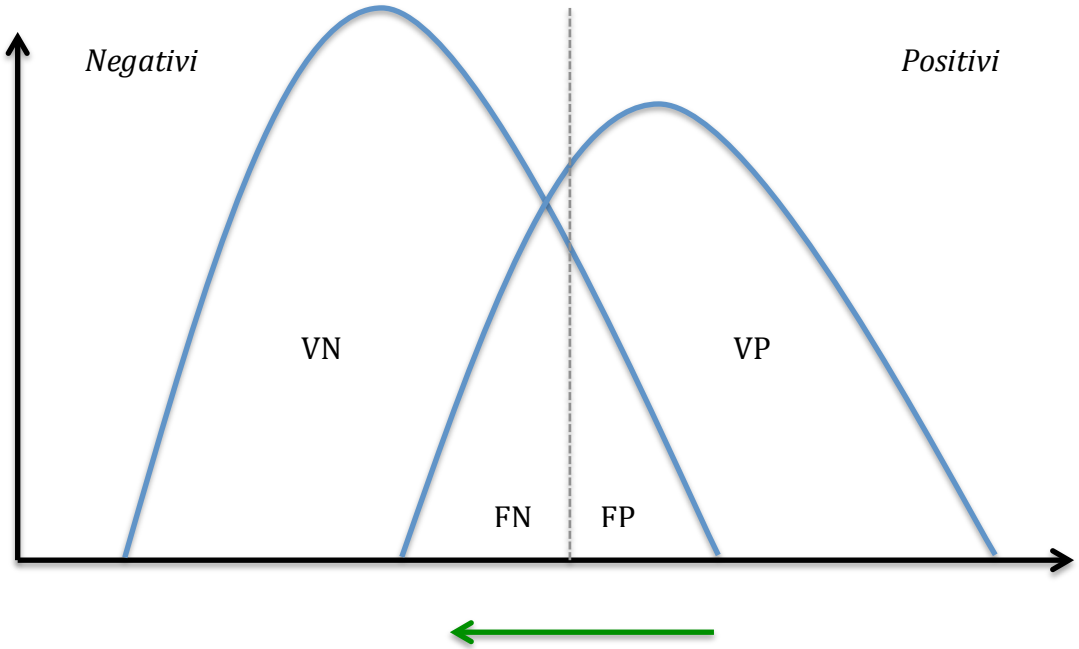
Le curve ROC (note come Receiver Operating Characteristic o Relative Operating Characteristic) sono funzioni grafiche di un classificatore binario. Lungo l'asse y è rappresentato il valore di sensitività, lungo l'asse x quello di $1 - specificità$, rispettivamente la frazione dei veri positivi e la frazione dei falsi positivi.
La Curva ROC studia quindi il rapporto tra allarmi veri e falsi allarmi, perché viene costruita tracciando il valore della frazione dei veri positivi rispetto a quella dei falsi positivi, in base a diversi valori di soglia.
E' una curva che infatti viene distribuita tra le due estremità (0,0) e (1,1).
La curva ROC può essere rappresentata come:




Ogni punto individuato dalla curva ROC rappresenta un diverso valore soglia, noto anche come cut-off. Ad ogni cutoff si calcola la frazione dei veri positivi sui falsi positivi e individua un punto sul grafico. All'aumentare della frazione dei veri positivi anche quella dei falsi positivi aumenta e, migliore è il modello di

previsione che stiamo applicando, più velocemente la frazione dei veri positivi si avvicinerà al valore 1 (100%).
Teoricamente il test perfetto restituirebbe una linea verticale retta da (0,0) a (0,1) pero poi orizzontalmente arrivare a (1,1).

Come e perché varia il valore di cutoff?
Se si considera un problema di predizione a due classi con classificatore binario, e scelto un valore di soglia rispetto a cui effettuare il test (cercando di classificare quindi le due classi), ecco che le due curve tendono  sovrapporsi, e scegliere un nuovo valore soglia significa "spostare le curve" e individuare il nuovo criterio che suddivide quindi in diversa proporzione i quattro risultati (della matrice di classificazione).
Ad ogni cutoff andremo a calcolare le proporzioni di interesse che, al cambiare del valore soglia) individueranno quindi la Curva ROC.




*Scelta del valore di cutoff*

### *2.7.3 Area Under the Curve AUC*

Un valore ricavabile dalla Curva ROC e utilizzabile come misura di accuratezza del test è AUC (Area Under the Curve). Un valore più alto di AUC indica tendenzialmente migliori performance del modello.
Il suo valore è una misura aggregata di tutti i possibili risultati di classificazione in base ai cut-off scelti, per una lettura più immediata può essere considerato la probabilità che il modello classifichi un elemento positivo più frequentemente di uno negativo.

Per stimare il valore di AUC si utilizzano generalmente due metodi. Il primo è empirico e nonparametrico, realizzato da DeLong, metodo divenuto popolare per non utilizzare le assunzioni della Distribuzione Normale.
Il secondo è realizzato da Metz e McClish e si base sugli studi della BiNormale

**AUC metodo DeLong**
Il calcolo AUC con il metodo DeLong è calcolato andando a sommare le aree dei trapezoidi che si formano tra i punti che connettono la Curva ROC.
DeLong definisce la componente $T_1$ dell'elemento i-esimo e individua:

$$V(T_{1i}) = \frac{1}{n_0}\sum_{j=1}^{n_0}\psi(T_{1i}, T_{oj})$$

e la componente $T_0$ del j-esimo elemento individuando:

$$V(T_{oj}) = \frac{1}{n_1}\sum_{i=1}^{n_1}\psi(T_{1i}, T_{oj})$$

la funzione $\psi(X, Y)$ è definita come:

$$\begin{cases} Y > X & \psi(X,Y) = 0 \\ Y = X & \psi(X,Y) = 1/2 \\ Y < X & \psi(X,Y) = 1 \end{cases}$$

L' *Area Under the Curve* sarà stimata come la somma dei valori $V(T_{1i})$ e $V(T_{oj})$ calcolati, e avrà quindi un valore compreso nell' intervallo (0,1)

$$AUC = \sum_{i=1}^{n_1} \frac{V(T_{1i})}{n_1} = \sum_{j=1}^{n_0} \frac{V(T_{oj})}{n_0}$$

La varianza di AUC è stimata come:

$$\sigma^2 AUC = \frac{1}{n_1} S_{T_1}^2 + \frac{1}{n_0} S_{T_0}^2$$

dove $S_{T_1}^2$ e $S_{T_0}^2$ sono calcolate come:

$$S_{T_i}^2 = \frac{1}{n_i - 1} \sum_{i=1}^{n_i} [V(T_{1i}) - AUC]^2$$

**AUC metodo McClish**

Supponiamo di avere due popolazioni, una composta da individui con la condizione osservata negativa e la seconda con individui con condizione osservata positiva, ci riferiremo ad esse rispettivamente con X e Y.
Si supponga che X e Y siano distribuite Normalmente, con una propria media e varianza:

$$X \sim N(\mu_x, \sigma_x^2)$$

$$Y \sim N(\mu_y, \sigma_y^2)$$

La curva ROC è tracciata con la seguente funzione:

$$\{FP(c), VP(c)\} = \left\{ \phi\left(\frac{\mu_x - c}{\sigma_x}\right), \phi\left(\frac{\mu_y - c}{\sigma_y}\right) \right\}, \quad -\infty < c < \infty$$

con c che sta ad indicare il valore di cut-off e $\phi(z)$ che rappresenta la funzione di distribuzione Normale.

L'area AUC può essere calcolata come:

$$AUC = \int_{-\infty}^{+\infty} VP(c)FP'(c)d_c$$

$$= \int_{-\infty}^{+\infty} [\phi\left(\frac{\mu_y - c}{\sigma_y}\right)\phi\left(\frac{\mu_x - c}{\sigma_x}\right)]\, d_c$$

$$= \phi[\frac{a}{\sqrt{1+b^2}}]$$

dove:

$$a = \frac{\mu_y - \mu_x}{\sigma_y} = \frac{\Delta}{\sigma_y}$$

$$b = \frac{\sigma_x}{\sigma_y}$$

E' possibile calcolare anche solo una porzione dell' area AUC, perché può interessare considerare solo un range della frazione dei falsi positivi, che corrisponde all' intervallo specifico di due cutoff $(c_1, c_2)$, in questo caso:

$$AUC = \int_{c_1}^{c_2} VP(c)FP'(c)d_c$$

# 3. ALBERI DECISIONALI

L'albero di decisione è un modello predittivo che rappresenta una serie di decisioni logiche, simili a un Flowchart, che si realizzano, come facilmente intuibile dal nome, in una struttura ad albero.
L'albero è composto da:

- nodi decisionali, ciascun nodo rappresenta una variabile
- archi decisionali che rappresentano un possibile valore di quella proprietà variabile e che partono dal nodo padre al nodo figlio
- foglie, che rappresentano il valore predetto per la variabile obiettivo presa in considerazione a partire dai valori delle altre variabili.

Il modello di previsione nell'albero è rappresentato dal cammino (path) dal nodo radice (root) al nodo foglia (leaf).
L'albero di decisione si presta per quelle applicazioni dove i meccanismi di classificazione necessitano di essere chiari e il risultato deve essere condiviso al fine di effettuare facilmente la decisione corretta. La facilità di lettura e la chiarezza dei dati esposti nel modello lo rendono una valida metodologia predittiva nel Machine Learning.
Ecco quindi che alcuni utilizzi includono:

- Modelli di Credit Scoring dove i criteri che stabiliscono la bontà o meno del creditore devono essere ben specificati
- Studi di marketing, quali attrazione e soddisfazione della clientela
- Studi di diagnosi per condizioni mediche basate su misurazioni di laboratorio

Questi sono solo alcuni degli esempi in cui gli alberi di decisione possono fornire informazioni di decisioni utili, e dal momento che possono essere utilizzati praticamente con qualsiasi tipologia di dato, sono tra le tecniche più largamente usate nel Machine Learning.

Potrebbe essere comunque utile analizzare qualche scenario dove gli alberi decisionali non rappresentano una scelta ottimale, come ad esempio un caso dove sono presenti un ingente numero di dati di tipo nominale con vari livelli o di contro un caso dove abbiamo un numero elevato di variabili, anche numeriche, poiché potrebbe tradursi in una struttura ad albero molto complicata, a seguito del vasto numero di decisioni, e di difficile lettura.

Gli alberi decisionali usano un' euristica chiamata Partizione Ricorsiva. Questo metodo è generalmente noto come *divide and conquer* perchè in base al valore della variabile suddivide i dati in sottogruppi via via più piccoli.
Cominciando dal nodo radice (root), che rappresenta l'intero dataset, l' algoritmo sceglie la variabile più accurata per ottimizzare la previsione della variabile target scelta, e i dati vengono quindi divisi in rami diversi a seconda delle specifiche richieste dalla variabile.
L'algoritmo effettua lo stesso procedimento ad ogni nodo fino a quando si raggiunge la dimensione prefissata o quando non ci sono più variabili per distinguere i dati ulteriormente.
Ci sono diversi algoritmi utilizzabili per la creazione di modelli di alberi decisionali, tra i più importanti C5.0 e CART, che si differenziano per caratteristiche quali: la scelta del miglior split, tecniche di tree pruning e la struttura dell' albero

**Scegliere il miglior Split**
La prima difficoltà per un albero di decisione è la scelta della variabile con cui effettuare la prima divisione del dataset, in quanto suddividere il dataset in base al valore di una variabile non significa riuscire a separare distintamente nei 2 rami le osservazioni con la variabile target di riferimento dal resto.
Ecco che interviene il concetto di purezza, ed è proprio la sua misura che caratterizza le differenti misurazioni per identificare il criterio con cui effettuare lo split. Le misurazioni di riferimento sono il calcolo dell' Entropia e l'Indice di Gini.

## *3.1 Entropia*

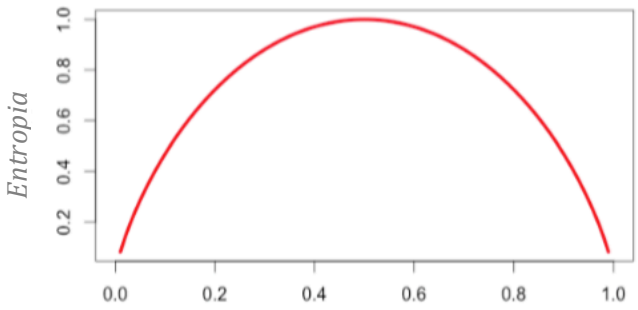
L'Entropia di un insieme di dati indica il livello di omogeneità che c'è tra le classi di dati, quindi un valore minimo pari a 0 indica che l'insieme dei dati è completamente omogeneo, mentre il valore massimo 1 indica il massimo disordine, inteso come la possibilità di trovare nella stessa classe quanti più dati con parametri diversi.
La definizione propria di Entropia è:

$$Entropia\ (S) = \sum_{i=1}^{c} -p_i log_2(p_i)$$

dove per un dato insieme di dati (S) il termine c indica il numero delle differenti classi e p si riferisce alla proporzione in cui i dati sono distribuiti nelle stesse. Un semplice esempio potrebbe essere quello di avere 2 classi di dati rispettivamente distribut3 al 60% e 40%, ecco che il calcolo dell'entropia diventa:

*-0.60*log2(0.60) - 0.40*log2(0.40)  pari a 0.9709506*

Come si nota è possile calcolare l'Entropia di un qualsiasi insieme insieme a 2 classi, se di uno si conosce la sua proporzione x, l'altra sarà 1-x.
La curva che descrive l'Entropia è la seguente:




Dal grafico si evince come il massimo valore dell' Entropia coincide con il valore x=0.50 (quindi con uno split 50-50 tra le classi) di contro il suo valore si riduce fino allo zero man mano che una classe aumenta rispetto all'altra.

Considerata questa grandezza come una delle misure utilizzate per calcolare la purezza, ci si chiede come venga utilizzata dall'algoritmo in un albero decisionale. Ad ogni split, l'algoritmo calcola il cambio di omogeneità che ne risulta in seguito allo split per ogni singola variabile. Il calcolo è noto come Information Gain, e il suo valore per una precisa variabile è nota come la differenza tra l'Entropia nel

punto precedente allo split (S1) e quella risultante in seguito allo split (S2)

*Information Gain (F)= Entropia (S1)- Entropia (S2)*

La complicazione di questo calcolo deriva dal fatto che a seguito di ogni split i dati sono divisi in più di una partizione, e la funzione deve quindi tener conto di tutta l'Entropia di tutte le partizioni ottenute, ecco che si associa un peso ($w_i$) al valore dell'Entropia ottenuta in base a quanti valori cadono in una specifica partizione.

In parole semplici, il totale dell'Entropia in seguito ad uno split è calcolata come la somma delle entropie delle singole partizioni *n* "pesate" in base alla proporzione dei dati che cadono in esse (Wi).

$$Entropia\ (S) = \sum_{i=1}^{n} w_i Entropia\ (P_i)$$

Maggiore è il valore dell' Information Gain, migliore sarà la variabile di riferimento nel creare gruppi omogenei dopo lo split, infatti un valore pari a zero implica che non c'è riduzione dell'entropia (S1~S2) mentre di contro un valore pari a 1 significa che in seguito allo split si è raggiunto un insieme di gruppi completamente omogenei (S2=0)

## 3.2 Indice di Gini

L'indice di Gini definisce la varianza totale per tutte le classi K, è una misura di impurità calcolata come:

$$I(t) = \sum_{i \neq j} p(i \text{ I } t) p(j \text{ I t})$$

La parte a destra dell'equazione rappresenta la stima di errata classificazione di un'osservazione di classe j nella classe i, quando si assegna casualmente un' unità del nodo t ad una particolare classe. L'impurità totale di un generico albero T è:

$$I(T) = \sum_{t \in \tilde{T}} I(t)p(t)$$

che esprime una variazione della funzione di impurità, difatti l'Indice di Gini si applica proprio in questo modo, cercando ad ogni split quello che massimizza il decremento di impurità ΔI

$$\Delta I(s^*, t) = I(t) - (I(t_L)p_L + I(t_R)p_R$$

Questa quantità è sempre positiva e assume valore zero nella situazione in cui :

$$p(j \text{ I } t_L) = p(j \text{ I } t_R) = p(j \text{ I } t) per\ j = 1, \dots, J$$

ovvero in una situazione di perfetta uguaglianza, di contro un valore pari a "1" indicherà una forte disuguaglianza. Lo split ottimale che massimizza il decremento di impurità ΔI è equivalente alla selezione dello split che minimizza l'impurità totale dll' albero, questo vuol dire che ad ogni split si individua il criterio di ottimizzazione locale di un albero di classificazione che corrisponde anche alla sua ottimizzazione totale.

## *3.3 Algoritmo C5.0*

Ci sono numerose implementazioni degli alberi decisionali, uno dei più noti è l'algoritmo C5.0, sviluppato da *J. Ross Quinlan* e diventato uno degli standard per

l'applicazione degli alberi di decisione, poichè le performance misurate sono risultate molto simili a quelle di metodologie più avanzate come le Neural Networks e le Support Vector Machines.
I punti di forza e di debolezza di questo algoritmo possono essere riassunti in:

| Vantaggi | Svantaggi |
|---|---|
| Algoritmo di classificazione che riesce a lavorare con più tipologie di dati e problemi | Possibilità di incorrere in campioni/stimatori distorti |
| Processo di apprendimento automatico che riesce a gestire grandi quantità di dati, numerici e non numerici | Se non ben gestito in fase di analisi, c'è il rischio di overfitting dei dati |
| Algoritmo utilizzabile su dati con piccoli o grandi training set | Problemi nel modellare relazioni non basate sul concetto di "divisione parallela ad assi" |
| Risultati facilmente interpretabili per alberi di piccole dimensioni | Piccole modifiche nel training set possono risultare in grandi differenze nei risultati di testing |
| Efficienza maggiore di altri modelli complessi | Alberi di grandi dimensioni risultano di difficile interpretazione |

## *3.4 Classification and Regression Trees (CART)*

L'algoritmo CART fa riferimento al modello introdotto da Leo Breiman nel 1984, comprende l' analisi di Alberi di Classificazione e di Regressione e ad oggi si identifica con l'analisi principale tramite Alberi di Decisione.

Gli Alberi di Classificazione e di Regressione sono tecniche di Machine Learning utilizzati per costruire modelli predittivi tramite Partizione Ricorsiva a partire da un insieme di dati, il risultato può quindi essere rappresentato graficamente

come un albero di decisione.
Gli alberi di classificazione sono utilizzati per la previsione della classe (discreta) cui i dati appartengono, mentre gli alberi di regressione vengono usati per la previsione di valori
casuali continui.

### *3.4.1 Alberi di Classificazione*

Lo scopo dell'analisi è quello di fornire un classificatore, una regola tramite cui si possa associare un'etichetta $j \in C$ alla variabile target cui stiamo applicando il modello di previsione. Questa soluzione è valida nei casi in cui la variabile risposta y non è continua, il risultato di un albero di classificazione è infatti quello di avere una variabile risposta di tipo nominale, o nel caso di una variabile continua, può risultare comodo categorizzarla.
Per costruire un classificatore si considera un campione di osservazioni

$$L = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

con (xi, yi) che indica le osservazioni sull' iesimo campione.
Il primo passo per misurare l'accuratezza di un classificatore è l'analisi della probabilità di misclassificazione (ovvero la probabilità di sbagliare la categoria di un'entità, in base al nodo dove questa ricade).
Data quindi un campione L e un'unità (x,y) la probabilità è espressa come

$$R(d) = P\{d(x) \neq y)$$

dove d() esprime la funzione del classificatore
Ci sono varie tecniche per stimare R(d):
-stima mediante test set: consiste nel dividere un campione in un sottoinsieme dedicato a costruire il modello di previsione (learning set), ed in un sottoinsieme (test set) i cui dati saranno applicati al modello per stimare R(d)
-stima mediante cross validation: utilizzabile nel caso si abbia un campione con molte osservazioni, è utilizzata per risolvere il problema dell'overfitting nei training-set, dove per overfitting si intende il rischio di sovradattamento durante il processo di apprendimento induttivo, quindi l'algoritmo individua delle false regolarità.

Il processo della cross validation (o K-fold Validation) consiste nel dividere il

dataset in k parti di uguali dimensioni, 1/k del dataset sarà utilizzato per il validation test, la restante (k-1)/k comporrà il training set.
Il processo si completa nel seguente modo: si avvia il processo di apprendimento induttivo sul training set (k-1)/k e si costruirà quindi l'albero decisionale, una volta realizzato si verifica l'accuratezza del modello inserendo come dati in input quelli del validation set.
Questo iter va ripetuto k volte, si ottengono quindi k alberi ciascuno con un proprio valore predittivo, l'albero con il valore predittivo più alto sarà quello definitivo usato nel modello poichè risente meno del problema dell'overfitting.

I passi necessari per la costruzione dell'albero sono:
-selezione degli splits
-chiusura del nodo (ottenendo quindi un nodo foglia) o continuare a dividerlo
-assegnazione per ogni nodo foglia di una modalità per y (la nostra variabile target)

Ad ogni nodo t, partendo dal nodo radice (root) si sceglie lo split che rende più omogenei i dati all'interno dei due nodi successivi, quindi cercando di ridurre l'Entropia a zero. Lo split ottimale si ottiene:
-definendo P(j | t), la probabilità che y=j (dove y=j indica che la variabile risposta assume il valore della j-esima modalità nominale o ordinale) tale per cui:

$$\sum_{j=1}^{J} P(j \text{ I } t) = 1 \qquad \forall t$$

-si definisce poi una misura di impurità per il nodo t, in un range (0,1) dove l'impurità maggiore si ottiene quando le classi hanno massima eterogeneità, e il valore minore (nullo) si ottiene quando il nodo contiene dati appartenenti ad una sola classe. L'indice di impurità utilizzato è quello di Gini, come precedentemente descritto

$$I(t) = \sum_{i \neq j} p(i \text{ I } t) p(j \text{ I } t)$$

-si definisce un insieme S di splits binari s, dove ad ogni nodo si interroga il dataset su una condizione di partizione, che scinde quindi il nodo t nei due nodi successivi

Nella metodologia CART, ad ogni nodo si analizza il valore di una sola variabile esplicativa tra quelle del dataset per effettuare lo split e ottenere i nodi successivi, si interroga la variabile a seconda che la stessa sia continua o categoriale, determinando lo split rispettivamente con le domande "x(t)>k" e "x∈S", dove:
-k intende il valore della variabile di riferimento scelto in modo che permetta lo split con il massimo decremento di impurità ΔI
-S è l'insieme dei valori nominali/ordinali che può assumere la variabile esplicativa.
In seguito all'interrogazione della variabile al nodo lo split dividerà le osservazioni nei 2 nodi td(destra) e ts(sinistra) nelle rispettive proporzioni Pd e Ps.
Le variabili vengono interrogate fin quando non è più possibile trovare un modo che ottimizzi la suddivisione del dataset e che non consenta quindi un ulteriore decremento di impurità. In questo punto si assegna un output, ovvero alla classe associata al nodo finale si assegna la modalità j (il valore di classificazione prescelto).
In che modo? Si assegna l'etichetta della variabile y più presente nel nodo, questo sottintende che nello stesso nodo non avremo variabili omogenee, variabili quindi con stessa modalità j. Da qui il calcolo della probabilità di misclassificazione R(t), che è una misura di accuratezza del modello adottato, si valutano cioè quanti valori sono finiti nel nodo "giusto", che esprime quindi la corretta modalità j per la variabile y.

Da sottolineare come se il numero degli split aumenta tendenzialmente si ha un valore più piccolo di R(h), perchè maggiore è il numero di interrogazioni alle variabili esplicativa, e maggiori sono le possibilità di suddividere il dataset, questa proprietà è valida poichè

$$R(t) \geq R(t_D) + R(t_S)$$

ovvero il nodo padre avrà sempre maggiore errore di misclassificazione rispetto ai nodi figli, poichè in seguito allo split necessariamente c'è un decremento dell'impurità.

### *3.4.2 Alberi di regressione*

Gli alberi di regressione sono utilizzati quando la variabile y è una variabile casuale continua. La differenza con la tipologia di albero precedente è che ora non si vuole più costruire una regola di classificazione, ma l'obiettivo è quello di stimare la relazione che intercorre tra la variabile y e il vettore di variabili indipendenti.
La regressione ad albero, come con il metodo di classificazione, cerca di suddividere nel modo più omogeneo possibile le unità che costituiscono il campione, partendo dal nodo radice ed effettuando una successione di split.
Lo split scelto al nodo h ,nell'insieme S degli split considerati, sarà quello che massimizza il valore:

$$\Delta R = R(h) - R(h_S) - R(h_D)$$

dove

$$R(h) = \frac{1}{N} \sum_{x_t \in h} (y_y - \bar{y}_h)^2$$

che è una grandezza che ricorda la misura della varianza, questo non sorprende considerato che come descritto anche per l'albero di classificazione il valore R è una misura dell'accuratezza del modello adottato, nel singolo nodo infatti N è pari al numero di osservazioni, mentre y(segnato) è la media dei valori della variabile y in h.

Si può quindi definire la funzione d'errore dell'intero albero H come:

$$R(H) = \sum_{h \in H} R(h) = \frac{1}{N} \sum_{h \in H} \sum_{x_t \in h} (y_t - \bar{y}_h)^2$$

## 3.5 Tree Pruning

I processi sopra descritti per alberi di classificazione e regressione possono produrre buoni risultati, ma uno degli aspetti da considerare quando si realizza un albero decisionale è anche il rischio di overfit dei dati, che come abbiamo visto in precedenza si può verificare in fase di processo di apprendimento induttivo.
Nonostante le considerazioni iniziali il risultato potrebbe essere comunque un albero di grandi dimensioni, quindi troppo complesso, con eccessiva varianza e di difficile interpretazione.
Un albero più piccolo, con meno split, e meno nodi finali, potrebbe risolvere queste problematiche, anche a costo di piccoli errori di distorsione sul campione.
Una buona strategia potrebbe quindi essere quella di realizzare con il nostro algoritmo il nostro albero, e se necessario, ridurre le dimensioni al fine di ottenere un "sottoalbero".
Per fare questo l'obiettivo è ottenere un sottoalbero che abbia una percentuale di errore con il campione di test quanto più minore possibile.
Il metodo della Cross Validation sarebbe applicabile, ma richiederebbe un calcolo eccessivo, poichè in questo caso andrebbe fatto per ogni singolo sottoalbero, e partendo dall' albero finale il numero di sottoalberi individuabile sarebbe elevato.
Si analizza quindi la funzione costo-complessità dell'albero, ovvero piuttosto che considerare ogni possibile sottoalbero si considera una sequenza di alberi indicizzati da un parametro non negativo α.
Per ogni valore di α si considera un sottoalbero individuato dall'albero finale tale per cui il valore

$$R_\alpha(H) = R(H) + \alpha H$$

sia il più possibile minore

α è il parametro positivo rappresentabile quindi come il trade-off tra la complessità del modello e la capacità di appossimare al meglio i dati. H è una misura della cardinalità dell'albero. Quando α =0 il sottoalbero coincide con l'albero finale e si ottiene lo stesso calcolo visto per l'albero di regressione, mentre se aumenta, c'è un costo da pagare per avere un albero con tanti nodi terminali e quindi con un valore H alto, di conseguenza il valore finale tenderà a diminuire con conseguente "potatura" dell' albero risultante quindi in un albero di dimensioni minori

## 3.6 Alberi di Decisione Vs Modelli Lineari

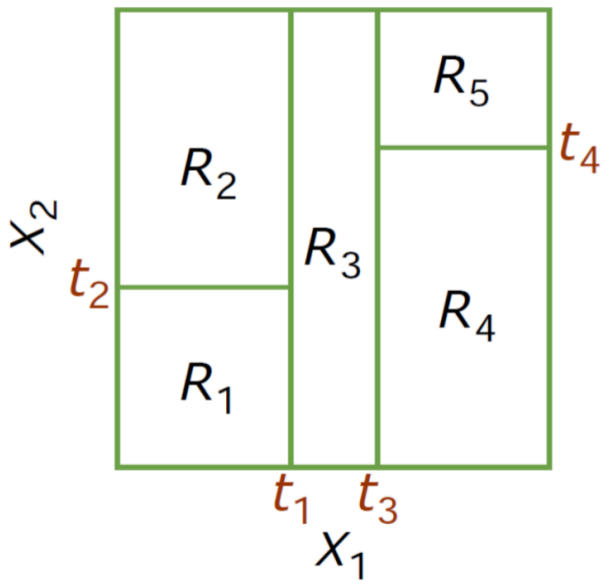
Gli alberi decisionali hanno un impostazione e caratteristiche diverse dai più comuni metodi di analisi lineare.
In particolare un modello di regressione lineare assume la forma:

$$f(X) = \beta o + \sum_{j=1}^{p} Xj\,\beta j$$

mentre ad esempio un modello generico di albero di regressione si esprime nella forma:

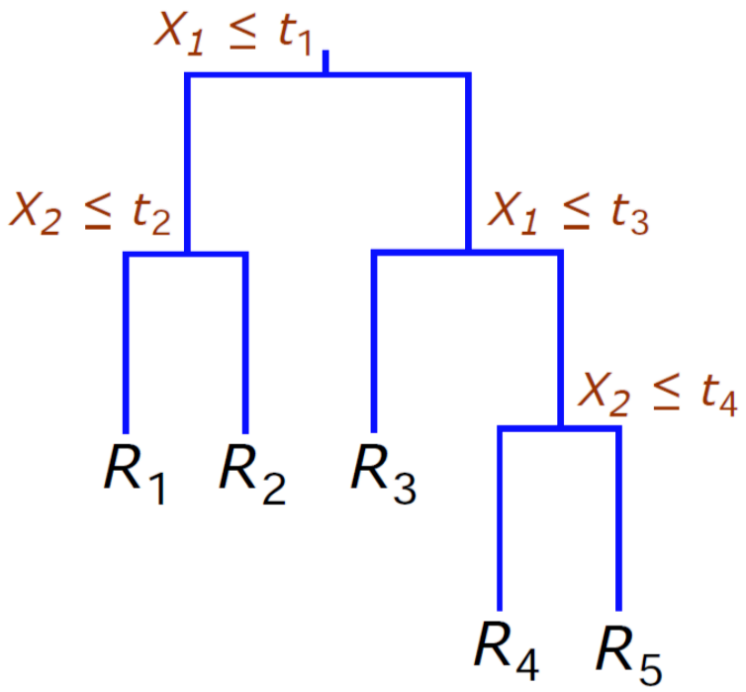
$$f(x) = \sum_{m=1}^{M} \left(c_m \cdot 1_{(X \in R_M)}\right)$$

dove R1,R2,...,Rm rappresentano le varie regioni in cui è possibile dividere lo spazio.
Cosa significa suddividere lo spazio in regioni?
Ogni qualvolta eseguiamo uno split in seguito ad un'interrogazione di una variabile per suddividere il dataset iniziale, (e cercare quindi di categorizzare la variabile target nel modo più omogeneo possibile) possiamo rappresentare la sequenza della partizione ricorsiva tramite diagramma.




Nella figura è mostrata la suddivisione in Regioni in seguito alla realizzazione di un modello di albero decisionale secondo le 2 variabili X1 e X2. In base a quelli

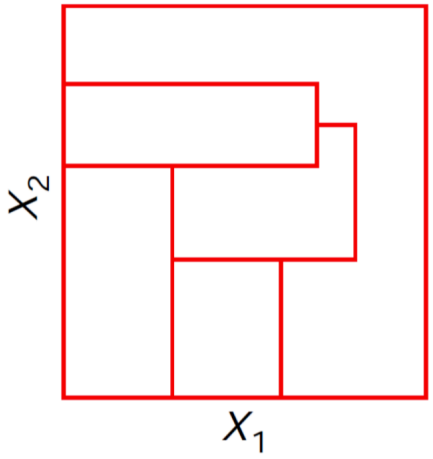
che sono i criteri per cui abbiamo deciso di dividere il dataset iniziale otteniamo le 5 regioni Ri che corrispondono ai 5 nodi finali del modello dell'albero analizzato. Per evidenziarne meglio il significato e ricondurlo a quanto analizzato nei paragrafi precedenti, possiamo rappresentare lo stesso diagramma tramite albero



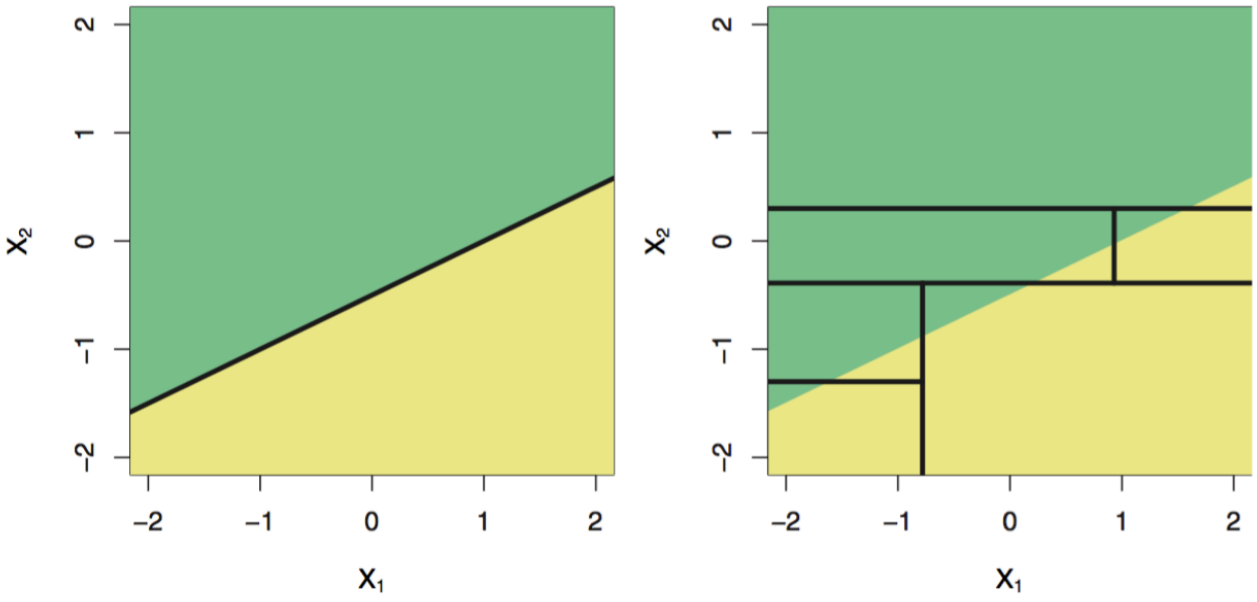

La seguente rappresentazione ha lo stesso significato della precedente, oltre a spiegare cosa si intende per regioni, possiamo anticipare un'altra differenza tra i modelli ad albero e quelli lineari ovvero, da quanto si evince nel diagramma di partizione delle regioni Ri, le linee di divisione (che rappresentano gli split) sono perpendicolari agli assi e non è un caso, infatti una caratteristica importante è che la partizione ricorsiva consista in split cosiddetti "paralleli agli assi", per ciascun criterio $t_i$ assegnato alla variabile $X_i$ (i=1,2) quindi si possono ottenere 2 nodi distinti, ciascuno che risponde al preciso criterio di riferimento

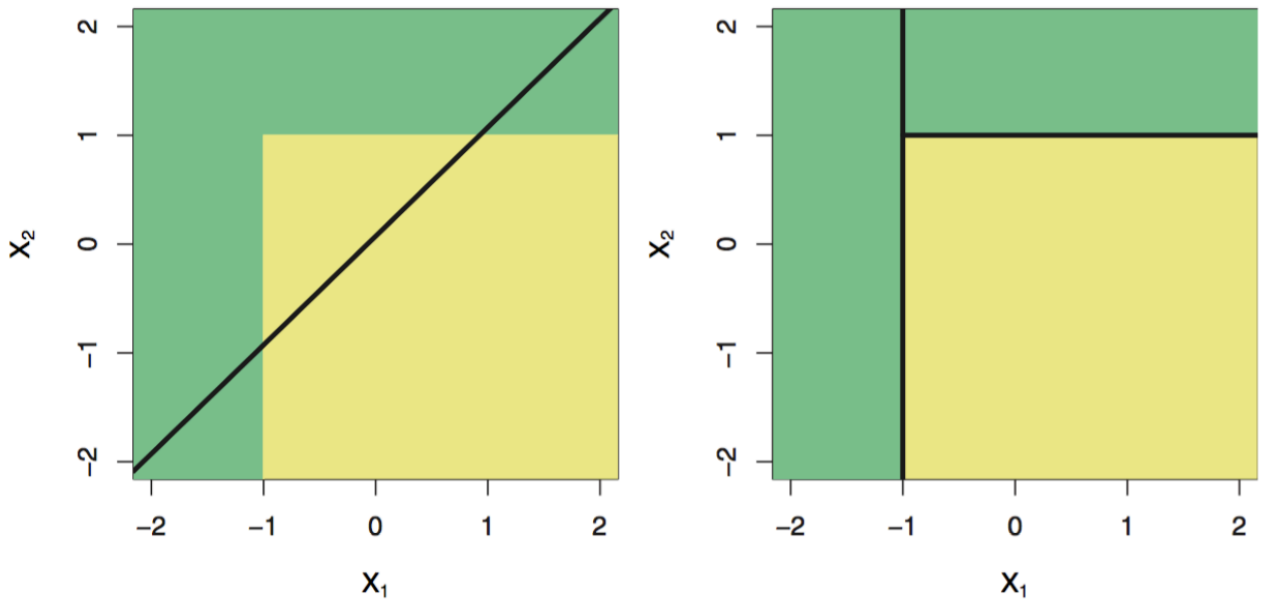
$$X_i \geq t$$
$$X_i \leq t$$

Un'altra caratteristica è che ad ogni split quindi deve necessariamente corrispondere una suddivisione precisa del dataset in 2 sottoinsiemi distinti, non sarà quindi possibile ottenere tramite partizione ricorsiva un diagramma del tipo:




Cosa scegliere tra modello lineare e modello ad albero?
Dipende dall'analisi del problema, se la relazione tra la variabile target e le variabili indipendenti può essere espressa dal modello lineare, ecco che la regressione lineare potrebbe esprimere bene questa relazione.
Qualora ci sia una relazione complessa, non lineare, ecco che un modello ad albero potrebbe misurare un'accuratezza ben maggiore nell'analisi dei risultati rispetto al modello di regressione lineare.
Di seguito un'analisi grafica di quanto espresso:






I diagrammi in alto mostrano come tra le variabili intercorra una relazione lineare (come si evince dalla divisione dell'area verde e gialla), vediamo quindi come un modello di regressione lineare (a sinsitra) riesca a suddividere la regione dello spazio in maniera decisamente più precisa rispetta a un modello ad albero(a destra).
Questo si traduce in un'analisi dei risultati più accurata con il modello di regressione lineare, sarà quindi possibile dividere il dataset iniziale in maniera più precisa, ottenendo 2 porzioni dello spazio ciascuna con alto livello di omogeneità, al contrario del diagramma a destra, dove si percepisce come l'errore di misclassificazione sia maggiore.

Al contrario i diagrammi in basso mostrano invece una relazione non lineare tra le variabili, applicando un modello ad albero riusciamo quindi a classificare meglio la variabile target, riuscendo a suddividere il dataset in maniera ottimale in base ai criteri espressi dalla relazione delle variabili.

Abbiamo visto come i modelli decisionali ad albero presentino vantaggi rispetto ai noti modelli lineari:
-sono facilmente esplicativi, la rappresentazione grafica aiuta
-l'approccio alla creazione di modello tramite albero decisionale, risulta più immediato nella comprensione
-Gli alberi possono gestire tutte le tipologie di variabili, qualitative e quantitative senza l'utilizzo di variabili dummy

Di contro non si dimostrano efficaci nel confronto con altri modelli qualora volessimo valutare l'accuratezza dei risultati o la robustezza dei dati, in quanto una piccola modifica nel dataset inziale potrebbe portare ad risultato finale anche molto diverso.

L'utilizzo degli alberi decisionali si è evoluto negli anni oltre gli algoritmi C5.0 e CART, per arrivare all'utilizzo di tecniche di Machine Learning quali Bagging, Random Forest e Boosting, che si traduce in una miglior performance predittiva.

## *3.7 BAGGING ,RANDOM FOREST, BOOSTING*

### *3.7.1 Bootstrap*

Gli alberi di decisione descritti fino ad ora possono produrre risultati con elevata varianza, questo significa che scegliendo dallo stesso campione un diverso training set e quindi un diverso set per il Testing, potremmo ottenere risultati anche molto diversi.
Da qui la necessità di utilizzare strumenti avanzati come Random Forest e Boosting, che producono risultati più consistenti e con minor varianza. Alla base di queste 2 metodologie c'è il bagging, ovvero aggregazione di dataset diversi ma derivati dallo stesso dataset iniziale tramite campionamento casuale.
Per descrivere al meglio queste tecniche di Machine Learning e lo tesso Bagging è necessario introdurre il concetto di Bootstrap.

Il *bootstrap* è una tecnica statistica di ricampionamento con reimmissione utilizzata per approssimare la distribuzione campionaria di una statistica.
Questa tecnica permette di calcolare l'errore standard e costruire intervalli di confidenza per varie tipologie di campioni.
Il metodo Bootstrap parte da un dataset iniziale, e tramite vari ricampionamenti crea nuovi dateset, ciascuno con proprie caratteristiche, quali ad esempio media e deviazione standard.
Il processo funziona in questo modo:
-La probabilità di assegnare un elemento dal dataset finale in uno dei dataset finali è la stessa per ogni elemento

-Il ricampionamento è con reimmissione, è possibile quindi trovare lo stesso elemento più di una volta nello stesso dataset finale
-La dimensione e il formato dei dataset finali sono gli stessi di quello iniziale.

La tecnica Bootstrap presenta diversi vantaggi rispetto ai metodi tradizionali, non è necessario fare assunzioni iniziali sulla distribuzione dei dati ed è possibile utilizzare questo metodo per un'ampia varietà di dati e anche con campioni di bassa numerosità; al contrario di altre metodologie che necessitano di assunzioni iniziali quali la distribuzione Normale o una numerosità minima per effettuare l'analisi.
Di contro è richiesta una certa complessità computazionale, in base alla "profondità" del calcolo, ed una conoscenza dei parametri con cui è possibile modellare la tecnica Bootstrap con i vari linguaggi di programmazione.

Proprio in R possiamo ottenere un semplice output che ci fornisce la distribuzione dei vari ricampionamenti, possiamo fare questo esempio con lo stesso dataset che utilizzeremo poi con le tecniche di Machine Learning, considerando le 2 variabili più significative del dataset per l'accuratezza dei vari modelli, ovvero I21e I19 (come vedremo nei prossimi paragrafi).

Utilizziamo la libreria *Boot*, che ci permette il ricampionamento e quindi l'utilizzo della tecnica Bootstrap

```
library(boot)

dis <- function(data, indices){
dt<-data[indices,]
c(
cor(dt[,22], dt[,20], method='s'),
median(dt[,22]),
median(dt[,20])
)
}
```

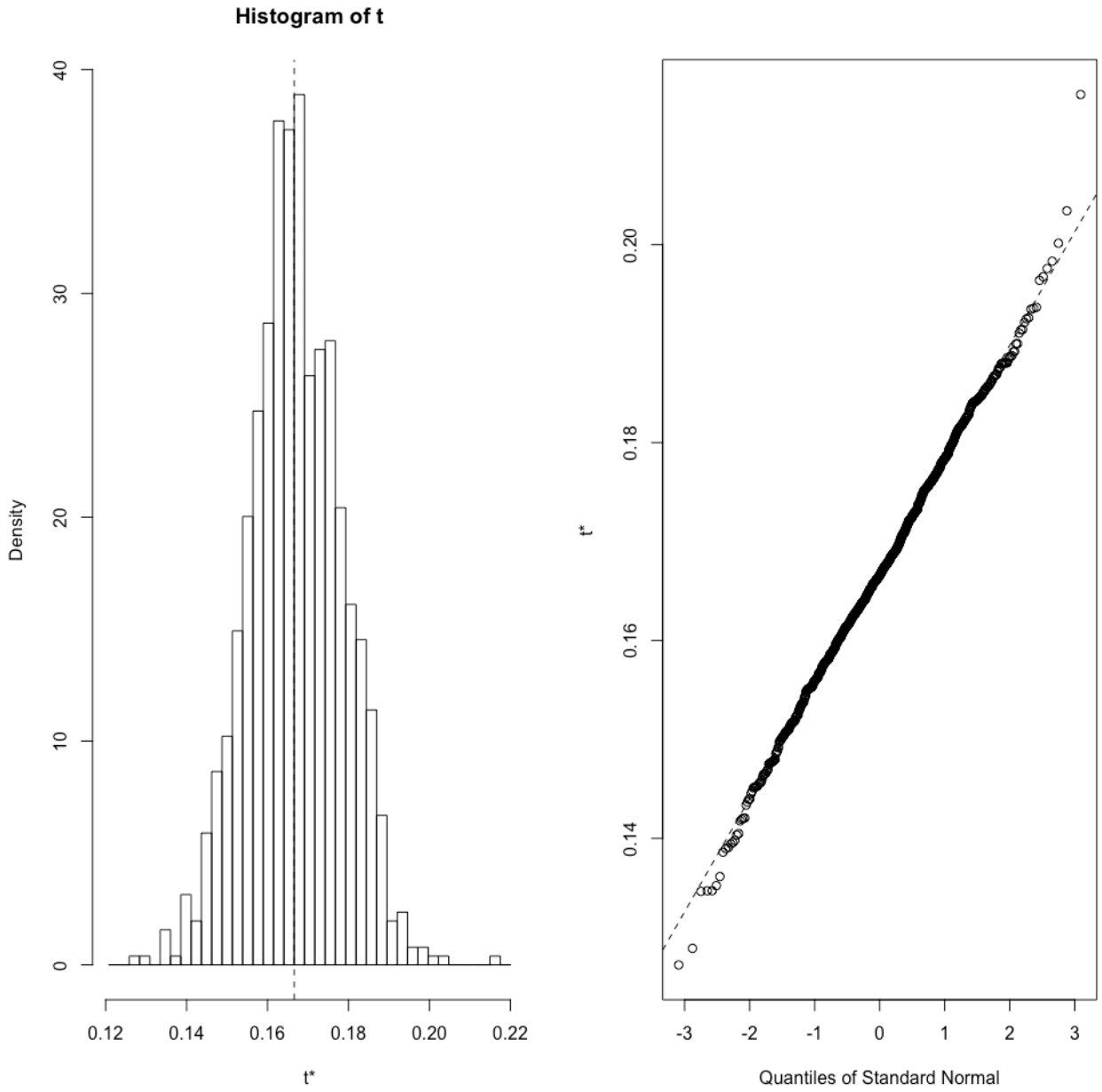
la funzione *dis* creata serve per assegnare gli elementi delle variabili I21 e I19 (che corrispondono alle colonne 22 e 20 del dataset iniziale) in un vettore, precisamente assegneremo la loro correlazione e le rispettive mediane.
Possiamo ora utilizzare la funzione Boot

```
set.seed(123)
Bootstrap <- boot(data, dis, R=1000, cor.type='s')
```

Il parametro *R=1000* serve appunto per assegnare il numero di ricampionamenti che la funzione deve restituire, otteniamo il seguente output, di cui sono interessanti le ultime 3 voci, che ci mostrano il risultato dei ricampionamenti da tecnica Bootstrap rispetto al dataset iniziale, infatti t1,t2,t3 sono riferiti rispettivamente ai valori della funzione, quindi la correlazione tra le 2 variabili e le loro mediane, con *Original* vediamo i valori del dataset iniziale, con *Bias* possiamo avere una prima misura della differenza tra le medie dei valori del dataset iniziale e quelle ottenute tramite Bootstrap.

```
ORDINARY NONPARAMETRIC BOOTSTRAP

Call:
boot(data = data, statistic = dis, R = 1000, cor.type = "s")

Bootstrap Statistics :
    original          bias          std. error
t1*  0.1665749  0.0003597381 0.011451743
t2*  0.9270219  0.0006003157 0.007111318
t3* 27.4700000 -0.0129524165 0.264771209
```

Eseguita la funzione possiamo quindi visualizzare la distribuzione ottenuta tramite tecnica Bootstrap con ricampionamento e reimmisisone tramite la funzione plot, e si nota come la distribuzione della correlazione tra le 2 variabili prescelte segua la distribuzione Normale.




### *3.7.2 Apprendimento Ensamble*

La tecnica Bootstrap appena esposta è alla base del Bagging, il cui nome deriva proprio dalla combinazione delle parole Bootstrap e *aggregation*.
Il Bagging è una tecnica di Machine Learning che rientra nella categoria dell'apprendimento ensamble, la cui caratteristica principale è l'utilizzo di una serie di metodi di insieme che usano modelli multipli per avere un' accuratezza maggiore e una miglior prestazone predittiva rispetto al modello originale.

I principali metodi dell' apprendimento ensamble sono:
-Bagging : crea un insieme di classificatori con la stessa importanza, ogni modello ottenuto viene valutato in base alla performance di predizione ottenuta e l'output complessivo sarà la classe che avrà ottenuto il maggior numero di voti.
-Boosting: ciascun classificatore avrà un certo peso sulla votazione finale del modello, e dipende dall'errore di ciascun modello in fase di learning

Il Random Forest è, come vedremo, un caso particolare del Bagging.

### *3.7.3 Bagging*

Partendo da un set con n osservazioni indipendenti N1,N2,…,Nn ciascuna con propria varianza $\sigma^2$ , posso ridurre proprio la varianza andando a calcolarla sul valore medio del dataset con N osservazioni, infatti la varianza di $\overline{N}$ risulta essere $\sigma^2/n$.

Di conseguenza per ridurre la varianza e migliorare l'accuratezza della predizione si prendono vari training set dalla popolazione originale e per ciascun training set si costruisce uno specifico modello per il testing set, e si calcola poi la media dei risultati predittivi.

Quindi, avendo un numero S di training set ed indicando con $f^S(x)$ il risultato predittivo di un specifico training set si può ottenere un modello con una varianza bassa andando a calcolare la media tra i vari modelli

$$f_{avg}(x) = \frac{1}{S}\sum_{s=1}^{S} f^s(x)$$

Il problema nell'ottenere un risultato del genere è la mancanza di un numero S di training set, non sempre è possibile e suddividere il dataset iniziale in troppi training set potrebbe comportare anche un risultato poco consistente.

Motivo per cui si applica la tecnica Bootstrap, si vanno cioè a creare training set secondo le modalità viste prima, ottenendo quindi S training set generati con

ricampionamento con reimissione su cui andiamo a costruire i modelli predittivi per calcolarne la media, ottenendo quindi:

$$f_{bootstrap}(x) = \frac{1}{S}\sum_{s=1}^{S} f^{*s}(x)$$

Questo è il principio alla base del bagging, e applicandolo ai modelli decisionali ad albero, è possibile migliorare l'accuratezza combinando centinaia o migliaia di alberi nello stesso modello per un'accuratezza maggiore e una varianza nei risultati minore.

Nel caso di modelli ad alberi decisionali, abbiamo due tipologie, Alberi di Regressione e di Classificazione. Nel primo caso, dove la variabile è quantitativa si può applicare direttamente quanto detto , nel caso di albero a classificazione la variabile target è qualitativa, ad esempio nel caso di modello predittivo per aziende a rischio default (come vedremo), la variabile binaria target non è quantitativa, ma identifica un'azienda sana da una anomala; in questo caso il modo più efficace (una volta ottenute le varie classi finali corrispondenti ai nodi finali) è quella di associare la variabile al valore target che predomina in una classe. In una stessa classe potremo infatti trovare aziende sane o anomale, l'importante è cercare di costruire un modello che tenda ad omogeneizzare il più possibile le classi, affinchè questo errore di misclassificazione sia il più possibile minore.

### *3.7.4 Stima dell'Errore OUT OF BAG*

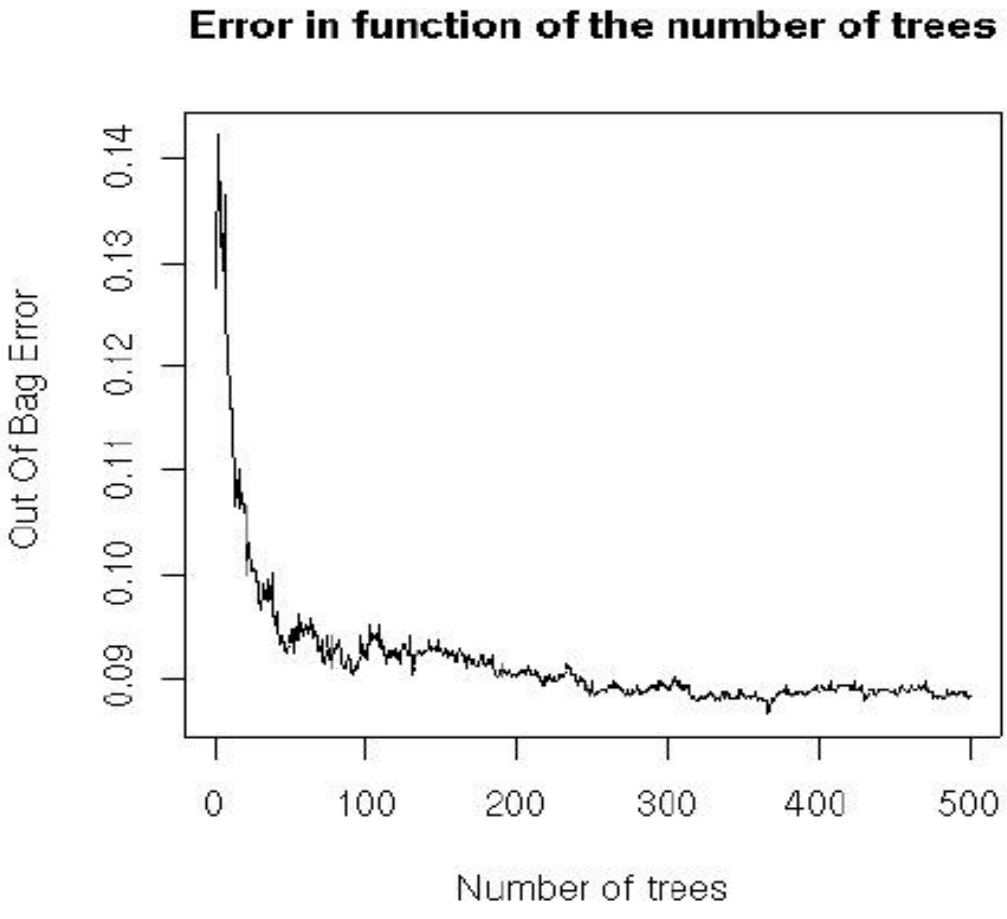
Abbiamo visto come il Bagging sia basato sulla tecnica Bootsrap, quest'ultima in particolare, quando applicata, tende ad utilizzare il 66% degli elementi della popolazione iniziale per la creazione dei diversi dataset, il restante 33% viene quindi utilizzato proprio per valutare l'accuratezza del modello e calcolare anche il livello di errore dell'albero finale. Le osservazioni che costituiscono questo 33% sono le cosiddette OOB (Out OF Bag observations).

Si vuole prevedere la risposta per l' iesima osservazione per tutti gli alberi in cui questa osservazione fa parte del set OOB. Dal momento che all'incirca il 33% degli elementi iniziali fa parte del set OOB, questo si traduce in un numero di predizioni pari all'incirca a N/3, dove N indica il numero totale delle osservazioni del dataset iniziale.

Anche qui otteniamo più predizioni per l' iesima osservazione, tante quanti sono

gli elementi OOB, per cui, come risultato finale, valuteremo la media dei risultati (nel caso di albero di regressione) o la numerosità maggiore per ogni classe (nel caso di albero di classificazione). Il risultato ottenuto è la predizione OOB per l'iesima osservazione, ma lo stesso ragionamento va applicato per tutte le n osservazioni, il risultato totale sarà quindi la stima dell'errore OOB che coincide con la stima dell'errore dell'intero albero cui è stato applicato il bagging.

La stima dell'errore finale, e quindi anche dall'accuratezza predittiva del modello, dipende dal numero di ricampionamenti effettuati per la creazione dei dataset finali, quindi dal numero di training set risultanti dal bagging e dal numero di alberi generati, tutti parametri che possiamo gestire per ottimizzare il modello finale, in base anche a quella che è la complessità computazionale richiesta. Un numero maggiore di alberi generati si traduce in una riduzione dell'errore, come si evince dal grafico seguente generato in R.



### *3.7.5 Random Forest*

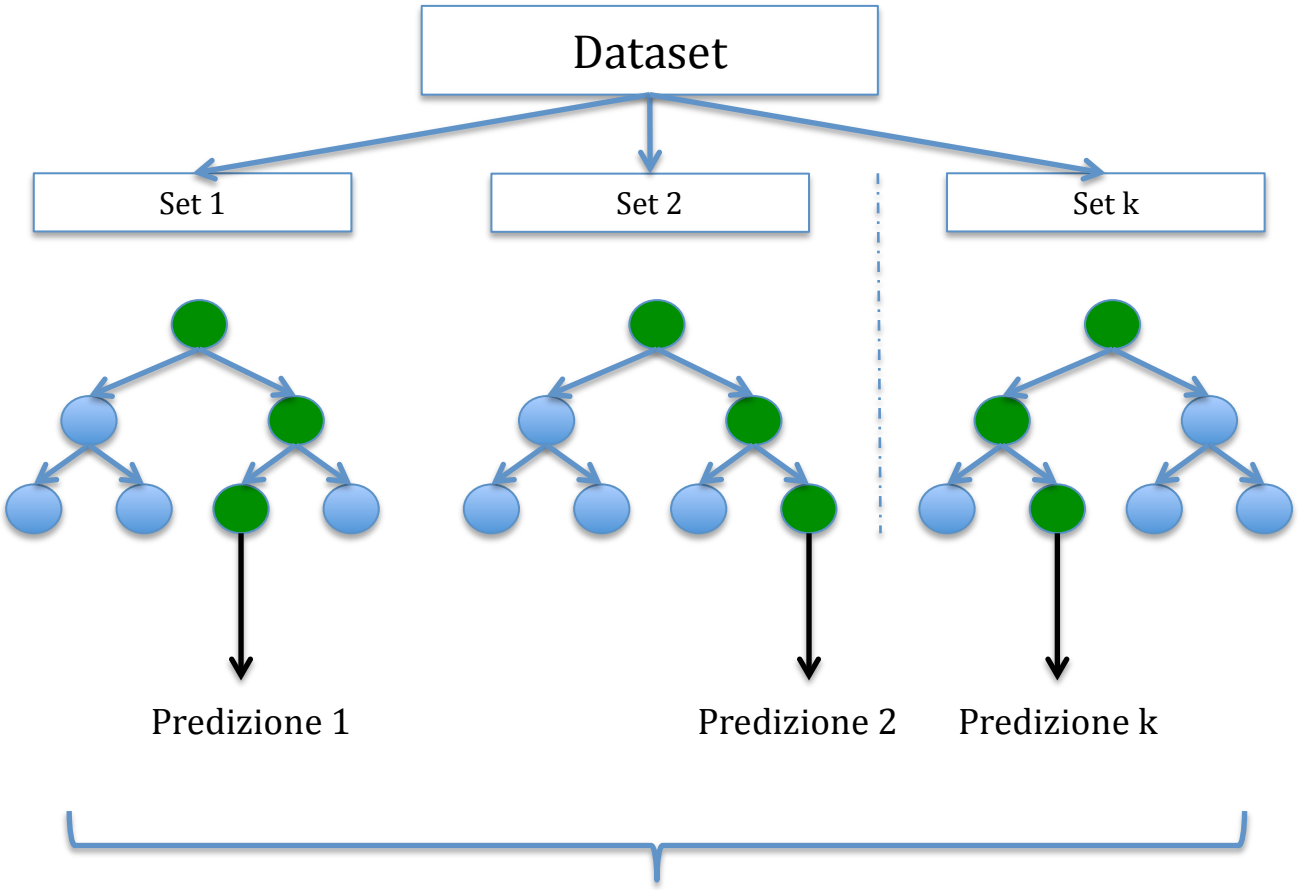
Il Random Forest è un caso particolare del Bagging, l'obiettivo è quindi sempre ridurre la varianza calcolando la media di molti modelli.
Anche per il Random Forest quindi si possono ottenere i migliori risultati con modelli basati su alberi decisionali, poiché questi catturano le strutture di interazione complesse presenti nei dati, che è difficile far emergere, e generando un numero elevato di alberi si ottiene tendenzialmente una distorsione bassa.
Nonostante i concetti alla base delle due tecniche siano i medesimi, con un corretto "tuning" dei parametri del modello si possono ottenere risultati migliori con la tecnica del Random Forest.
Questo perché gli alberi sono soggetti ad errore, il vantaggio quindi di usare il Bagging risiede nel poter generare alberi provenienti da una distribuzione identica, per cui il valore atteso di ciascuno sarà uguale al valore atteso di una media di N alberi generati.
Questo vuol dire che la distorsione dei modelli "bagged" è la stessa dei singoli alberi, per cui il miglioramento si può ottenere riducendo la varianza, ed è qui che interviene la particolarità dei modelli Random Forest, si cerca di migliorare la varianza del Bagging riducendo la correlazione tra gli alberi.
Questo si nota prendendo in esempio N variabili casuali indipendenti e identicamente distribuite, ciascuna con varianza $\sigma^2$, e con varianza della media su N variabili pari a $(1/N)\sigma^2$
Se le variabili però sono semplicemente identicamente distribuite e quindi non necessariamente indipendenti, supponendo una correlazione a coppie ρ positiva, la varianza della media è:

$$\rho\sigma^2 + \frac{1-\rho}{N}\sigma^2$$

Se N aumenta, si nota come il secondo termine diminuisce mentre il primo no, quindi se c'è alta correlazione non si riesce a ridurre la varianza.
La tecnica di Random Forest per ridurre la correlazione tra gli alberi e quindi la varianza consiste nel far crescere gli alberi attraverso la selezione casuale delle variabili di input.
La grande differenza rispetto al Bagging consiste quindi nel selezionare un numero di variabili candidate per lo split minore rispetto a quelle totali.
Dopo aver fatto crescere S alberi, il predittore basato sul modello di Random Forest sarà:

$$f_{rf}^{S}(x) = \frac{1}{S}\sum_{s=1}^{S} T_s(x;\varphi)$$

dove $s_i$ indica l'albero del modello di Random Forest e il termine $T_s(x;\varphi)$ è la previsione dello stesso albero $s_i$.
L'algoritmo di Random Forest può essere riassunto in questi step:
-Per *s = 1…S*
• Si estrae un campione tramite tecnica Bootstrap di dimensione N dal training set
• Con i dati ottenuti tramite Bootstrap si genera un albero $T_s$ , si ripetono i passaggi per ciascun nodo terminale dell'albero
  ○ Si selezionano *m* variabili tra le *p* variabili totali a disposizione
  ○ Si sceglie la miglior variabile per ogni nodo a seconda delle condizioni di split
  ○ Si effettua lo split ottenendo due nodi
-Si ottiene l'insieme degli alberi ${T_b}_1^B$
-La predizione del modello si ottiene pesando le singole predizioni degli alberi ottenuti




### *3.7.6 Boosting*

Il Boosting è un altro approccio applicabile agli alberi decisionali per migliorarne le capacità di previsione. Come il Bagging anche il Boosting può essere applicato a modelli statistici di regressione e classificazione.
Come visto nei precedenti paragrafi Il Bagging è un processo che consiste nel creare diversi set a partire dal training set originale tramite tecnica Bootstrap, modellando un albero decisionale per ogni set, e combinare tutti gli alberi ottenuti per ottenere un singolo modello di predizione. L'aspetto su cui porre attenzione è l'indipendenza di ciascun albero generato da un set *Si* (i=1,…,K) rispetto ad un altro.

La particolarità del Boosting consiste invece proprio nel costruire gli alberi in sequenza, ovvero ciascun albero è costruito in base alle informazioni elaborate e ottenute dall'albero precedente.
A differenza di altre tecniche dove si cerca di modellare un albero di decisione sull'intero dataset, o comunque su un intero training set, l'obiettivo del Boosting è quello di ottenere un modello di predizione con "apprendimento lento".
Si parla di "apprendimento lento" poiché dapprima si crea un primo modello dei dati del training set, in seguito viene creato un secondo modello che tenta di correggere gli errori del primo, quindi piuttosto che costruire un nuovo modello basato sulla predizione Y precedente, si costruisce l'albero successivo cercando di modellare al meglio gli errori ottenuti. L'idea che ne segue è quella di formare sequenzialmente dei *weak learner*, ognuno che cerca di correggere gli errori del primo.
Ognuno di questi alberi può anche avere pochi nodi terminali, nonstante il dataset iniziale sia ampio, questo può dipendere sia dai parametri dell'algoritmo, sia dal fatto che piccoli alberi possono performare meglio gli errori dell'albero precedente.
L'obiettivo è, albero dopo albero, quello di migliorare il modello finale proprio dove non performa al meglio.

La tecnica di Boosting ha tre parametri su cui è possibile agire per modellare al meglio gli alberi decisionali:
- il numero di alberi T, come le tecniche di Bagging, a un numero più alto di T corrisponde una maggior accuratezza nella predizione, ma il rischio di overfitting è tendenzialmente più alto rispetto, ad esempio, alla tecnica Random Forest.
- un parametro $\theta$ che indica il grado di apprendimento dell'albero, è un numero solitamente basso, valori tipici sono 0.01 e 0.001, e ha una grande influenza sul risultato, un valore $\theta$ troppo basso richiede un numero di alberi T alto per avere un modello con buone performance.
- il numero s di split, di default il valore è pari a uno, si ottengono infatti i cosiddetti *stumps,* ovvero alberi con un solo nodo padre e due nodi foglia. Un valore s pari a uno è spesso utilizzato, rende il calcolo più veloce ma la sua modifica può cambiare nettamente il risultato.

E' possibile descrivere il Boosting con il seguente algoritmo:

- Si imposta una prima funzione di approssimazione del modello $\hat{f}(x)=0$
- Per t=1,…T :
• Si costruisce un albero $\hat{f}^t$ con un numero di split $s$ per modellare il training set
• Si aggiorna $\hat{f}(x)$ aggiungendo un primo weak learner

$$\hat{f}(\mathrm{x}) = \hat{f}(\mathrm{x}) + \theta \hat{f}^{t}(x)$$

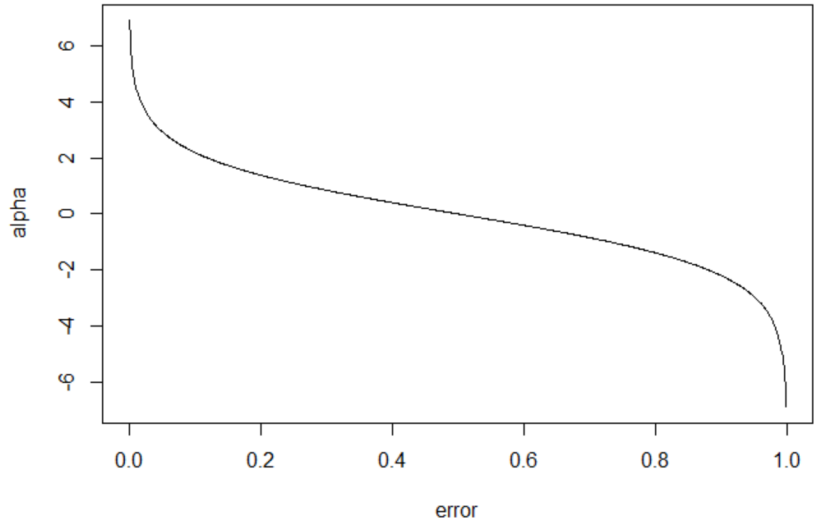
• Si aggiornano gli errori generati dal primo albero

$$r_i = r_i - \theta \hat{f}^{t}(x_i)$$

- Si genera l'output di Boosting

$$\hat{f}(\mathrm{x}) = \sum_{t=1}^{T} \theta \hat{f}^{t}(x)$$

Oltre la creazione iterativa degli alberi (quindi ogni albero dipende dal risultato del precedente) , un'altra particolarità che contraddistingue questa tecnica di Machine Learning è il peso diverso che si assegna a ciascun albero in base all'errore di misclassificazione che genera, in particolare maggiore è l'errore maggiore sarà il peso assegnatoli.




Ecco che l'output finale andrà corretto per la grandezza $\alpha$ che misura appunta il peso di ogni predizione, ottenendo:

$$\hat{F}(\mathrm{x}) = \sum_{t=1}^{T} \theta \hat{f}^{t}(x) * \alpha_{t}$$

L'obiettivo è minimizzare la funzione

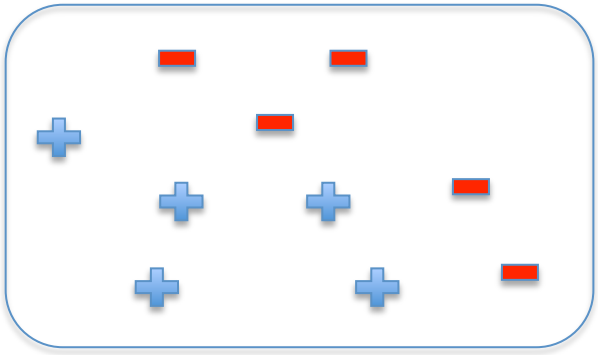
$$O(x) = \sum_{i} l(\hat{y}_i, y_i) + \sum_{t} \phi(f_t)$$

dove $l(\hat{y}_i, y_i)$ esprime la loss function, la distanza quindi tra il valore effettivo e il valore della predizione per l'elemento *i*

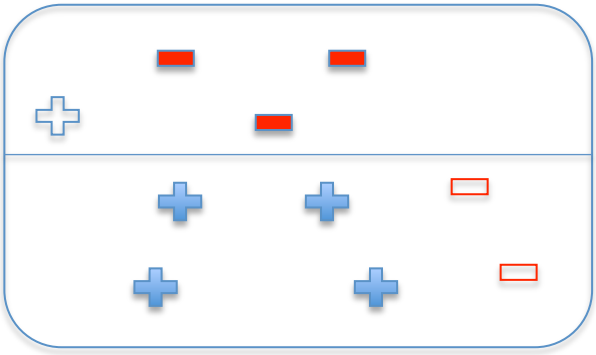
e $\phi(f_t)$ è una funzione che esprime il tradeoff costo-complessità della funzione, penalizza quindi la complessità dell'albero *t.*

Di seguito una rappresentazione grafica del modello di Boosting, dove ogni volta che si utilizza un classificatore si identificano gli elementi che "cadono" nella regione sbagliata, e li si associa ad un peso maggiore in modo che la combinazione dei classificatori utilizzati restituisca le regioni il più omogenee possibile.

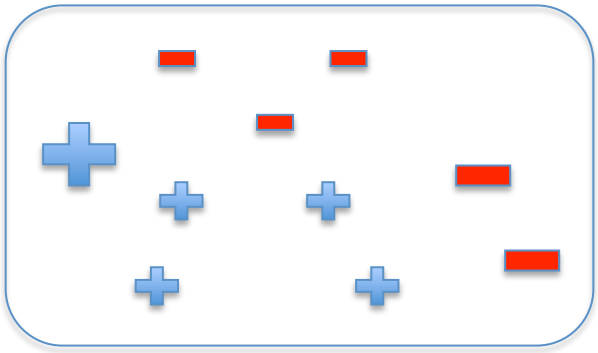
*Dataset originale D1*



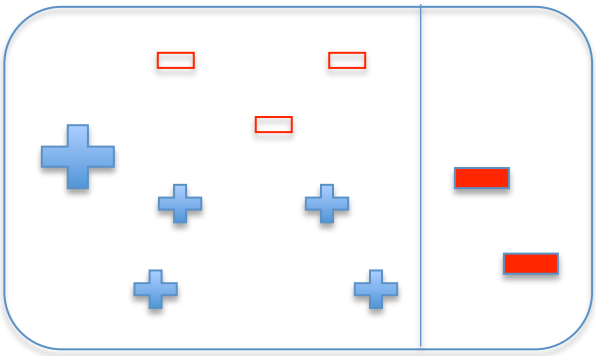
*Utilizzo primo classificatore*



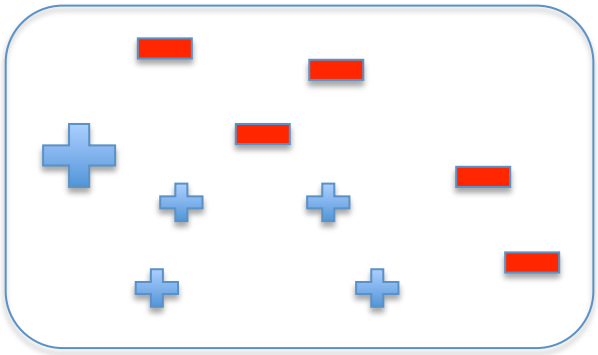
*Aggiornamento peso errori*



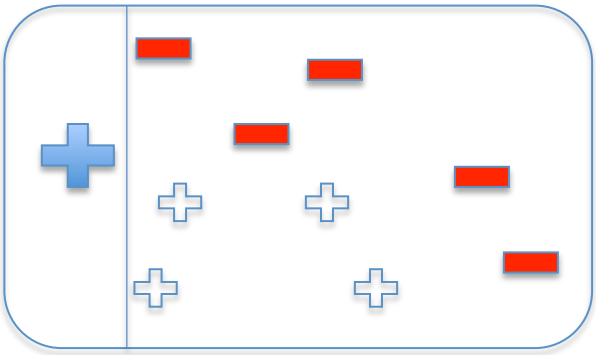
*Utilizzo secondo classificatore*



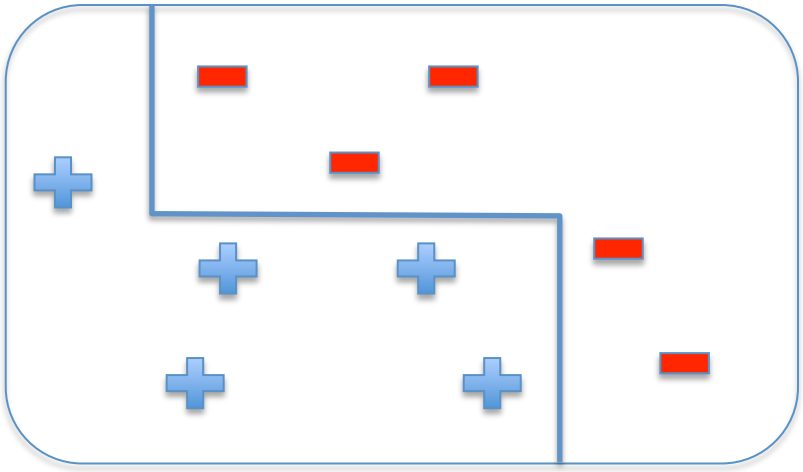
*Aggiornamento peso errori*



*Utilizzo terzo classificatore*



**Classificatori combinati**



Quanto appena descritto rappresenta il metodo Adaptive Boosting, meglio noto come **AdaBoost**, la tecnica si è evoluta poi negli anni, migliorando nella metodologia, sino ad arrivare ad algoritmi più noti come Gradient Boosting e XGBoost.

### *3.7.7 Gradient Boosting*

Il Gradient Boosting è una tecnica di Machine Learning che ha alla base la stima iterativa di alberi sui residui ottenuti ad ogni passo e l'aggiornamento in maniera adattiva delle stime. Questa tecnica riprende il concetto matematico del Gradient Descent, per cui lo split scelto sarà quello che favorisce l'avvicinamento al punto di minimo della funzione obiettivo.

**Gradient Descent**
Il Gradient Descent è un algoritmo di ottimizzazione che consente di individuare il valore minimo di una funzione di costo per sviluppare un modello con una previsione accurata.
Per minimizzare la Funzione di costo dobbiamo trovare quei parametri, i cosiddetti *weights* per cui la funzione abbia valore minimo, ecco perché si parla di gradiente, vogliamo mantenere un gradiente negativo, vogliamo raggiungere il punto di minimo.
Il ragionamento dietro questo algoritmo parte con la definizione di due componenti: la direzione verso cui stiamo cercando e il passo, cioè quanto lontano vogliamo andare in quella particolare direzione.
Si sceglie un punto di partenza, un punto arbitrario $x_0$ e ad ogni passo $k$ ci si muove in una direzione $\Delta x_k$ con un passo $t_k$ fino al prossimo punto $x_{k+1}$. Nel Gradient Descent cerchiamo il punto in cui il gradiente diventa negativo, dove quindi:

$$\Delta x = -\nabla f(x)$$

Questa ricerca iterativa può essere descritta da questa regola ricorsiva:

$$x_{k+1} = x_k - t_k \nabla f(x_k)$$

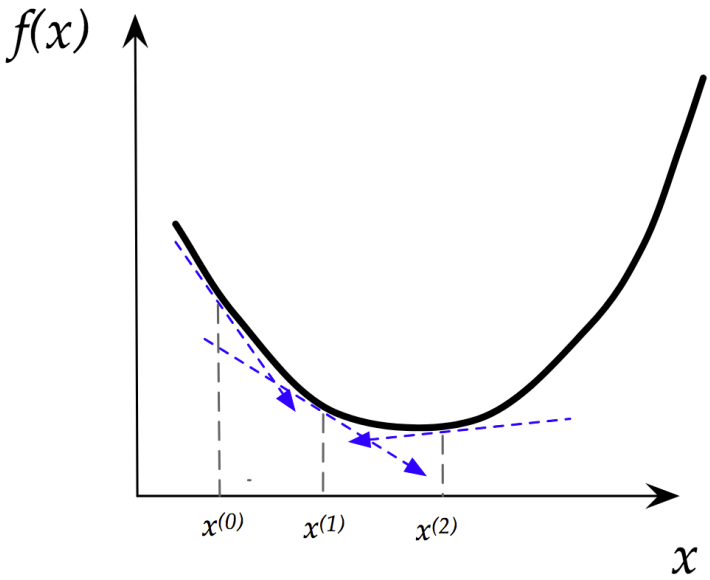
Come scegliamo $t_k$? Dal momento che il nostro obiettivo è quello di minimizzare la funzione, potremmo scegliere $t_k$ in modo da minimizzare il valore del prossimo punto $x_{k+1}$. Dal momento che $x_{k+1} = x_k - t_k \nabla f(x_k)$ il valore ottimale per $t_k$ è:

$$t_k^* = argmin_{t \geq 0} f(x_k - t \nabla f(x_k)$$

e possiamo descrivere l'algoritmo del Gradient Descent nel seguente modo:

Si sceglie $x_0$
While $\nabla f(x_k) \neq 0$ :
$x_{k+1} = x_k - t_k \nabla f(x_k)$
$k = k + 1$
end

L'algoritmo continuerà quindi fino a quando $\nabla f(x_k)$ assumerà verso negativo, che concettualmente indica un cambio direzione e quindi ci troviamo nell' intorno del punto di minimo che stavamo cercando.



**Problemi di Regressione con Gradient Boosting**

L'algoritmo Gradient Boosting per problemi di Regressione può essere descritto nei seguenti step:

**Input**: si considera il Training Set e il metodo che utilizzeremo per modellare al meglio i dati,
partiamo quindi dal dataset iniziale:

$\{(x_i , y_i)\}_{i=1}^{n}$

dove $x_i$ intende l' insieme dei valori delle variabili utilizzate per fare una predizione sulla variabile $y_i$.
Stabiliamo ora una Loss Function, ovvero una funzione differenziabile che esprima una valutazione della nostra predizione:

$$L(y_i, F(x))$$

una delle Loss Function più utilizzate per la Regressione con Gradient Boosting è:

$$\frac{1}{2}\ (Osservazione - Predizione)^2$$

che ricorda la Loss Function della regressione lineare

**step1**
Si inizializza il modello con un valore ricavabile da:

$$F_0(x) = \ argmin_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$$

dove $y_i$ si riferisce ai valori osservati e $\gamma$ ai valori di predizione ottenuti, la sommatoria indica che consideriamo una Loss Function per ogni $y_i$, e l'argomento $argmin$ indica che stiamo cercando un valore di predizione che minimizzi questa somma, in altre parole se rappresentassimo i valori $y_i$ su una retta vogliamo trovare il punto che minimizza la somma degli errori quadratici.
Questo si traduce nella media delle osservazioni $y_i$ , valore con cui inizializziamo il nostro modello.

**Step 2**

Scegliamo il numero di alberi M che vogliamo generare, e per m=1,…, M:
Calcoliamo

$$r_{im} = -[\frac{\delta L(y_i, F(x_i))}{\delta F(x_i)}]_{F(x)=F_{m-1}(x)}$$

Questa formula altro non è che la derivata della Loss Function scritta sopra, che con segno negativo risulta quindi essere

*= (Observed – Predicted)*

ovvero semplicemente il residuo, da notare inoltre come $F_{m-1}(x)$ per m=1 risulti proprio essere il valore trovato al punto 1, quindi semplicemente il valore con cui abbiamo inizializzato il modello, ovvero la media delle osservazioni $y_i$.
Il risultato del punto 2 risulta quindi essere il residuo $r_{im}$ ottenuto dalla prima osservazione *i* e per il primo albero *m*.

Piccola nota: la derivata $[\frac{\delta L(y_i, F(x_i))}{\delta F(x_i)}]$ è il gradiente da cui la tecnica Gradient Boosting prende il nome.

**Step 3**

Modelliamo un albero di regressione per il residuo e creiamo le regioni $R_{jm}$ , dove *m* indica l'abero che stiamo creando, il valore *j* indica il nodo foglia ottenuto.
Per J=1,…, $J_m$ calcoliamo:

$$\gamma_{im} = argmin_{\gamma} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$$

La grandezza $\gamma_{im}$serve per ottenere un unico valore come Valore di Output , questo perché nella regione $R_{jm}$, come abbiamo visto nel paragrafo degli alberi decisionali, possono cadere più osservazioni.
L'equazione è molto simile a quella già vista nel Punto1, la differenza è che ora consideriamo la Previsione ottenuta nell' albero precedente $F_{m-1}(x_i) + \gamma$.

L'argomento della sommatoria $x_i \in R_{ij}$ indica invece come nella sommatoria consideriamo solo le osservazioni $x_i$ che sono cadute nella rispettiva Regione $R_i$.

Considerata la nostra Loss Function

$$L(y_i, \gamma) = \frac{1}{2} (Osservazione - Predizione)^2$$

i valori di Output saranno sempre ottenuti come media degli errori residui che cadono nella stessa Regione (stesso nodo foglia)

**Step 4**

L'algoritmo si conclude con il calcolo della predizione per ogni osservazione:

$$F_m(x) = F_{m-1}(x) + \theta \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$$

Importante notare come la predizione $F_m(x)$ sia influenzata dalla predizione calcolata nello step precedente $F_{m-1}(x)$, a cui poi va ad aggiungersi l'albero appena creato.
Il parametro $\theta$ indica il grado di apprendimento dell' albero, come già abbiamo visto nel paragrafo introduttivo del Boosting.

**Problemi di Classificazione con Gradient Boosting**

La tecnica di Gradient Boosting può essere utilizzata anche per problemi di classificazione, gli step dell'algoritmo sono gli stessi ma ci sono delle considerazioni da fare sulla diversa tipologia di predizione e sulla Loss Function che inevitabilmente cambierà rispetto ai problemi di regressione.

**Input**
I dati in input saranno sempre $\{(x_i\ , y_i)\}_{i=1}^{n}$ , quindi l' insieme dei valori delle variabili $x_i$ utilizzate per fare una predizione sulla variabile $y_i$.

Ricaveremo la Loss Function partendo dalla Regressione Logistica( che ben si sposa con la natura della variabile di tipo dicotomico da prevedere), e partendo dal calcolo del *log(likelihood)* sui dati considerando le osservazioni $y_i$.

$$-\sum_{i=1}^{N} y_i * \log(p) + (1 - y_i) * \log\,(1 - p)$$

dove p si riferisce proprio alla probabilità che $y_i$ assuma valore pari a 1.
Calcolando il *log(likelihood)* e considerato la natura della variabile $y_i$ si nota come i termini utilizzati nella sommatoria siano solo *log(p)* per $y_i = 1$ e *log(1-p)* per $y_i = 0$

L'equazione appena vista deve essere trasformata, in modo che sia una funzione dei *log(odds)* di $y_i$, invece che di *p.*

Riscriviamo l'equazione con i seguenti passi
- $-y_i$ * log( p ) – (1 - $y_i$) $* \log(1 - p)$
- $-y_i * \log(\ p) - \log(1 - p) +\ y_i * \log(1 - p)$
- $-y_i * [\log(p) - \log(1 - p)] - \log(1 - p)$

visto che $\log(p) - \log(1 - p) = \frac{\log(p)}{\log(1-p)} = \log\left(\frac{p}{1-p}\right) = \log(odds)$

- $-y_i * \log(odds) - \log(1 - p)$

ora convertiamo log (1-p) in una funzione di log(odds) , e tramite metodo della massima verosimiglianza troviamo

$\log(1 - p) = \log(1 - \frac{e^{\log\,(odds)}}{1+e^{\log\,(odds)}}) = \log\,(\frac{1}{1+e^{\log\,(ods)}})$=$\log(1) - \log\left(1 + e^{\log(odds)}\right)$

quindi $\log(1 - p) = -\log\,(1 + e^{\log(odds)})$

- $-y_i * \log(odds) +\ \log\,(1 + e^{\log(odds)})$

dal momento che $y_i$ sono le osservazioni, possiamo riscriverla come

- $Osservazione * \log(odds) + \log\,(1 + e^{\log(odds)})$

verifichiamo che sia differenziabile

$$\frac{\mathrm{d}}{d\log(odds)} = -Osservazione + \frac{1}{1+e^{\log(odds)}} * e^{\log(odds)} =$$

$$= -Osservazione + \frac{e^{\log(odds)}}{1+e^{\log(odds)}} = -Osservazione + p$$

**Step 1**

Il valore con cui inizializzare il modello è sempre:

$$F_0(x) = argmin_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$$

dove $L(y_i, \gamma)$ si riferisce alla Loss Function appena calcolata, il ragionamento è lo stesso, ma con i problemi di classificazione il valore sarà pari a p, ovvero la probabilità che $y_i$ assuma valore pari a 1.

Quindi il valore *log(odds)* per gamma che minimizza la somma è

$$F_0(x) = \log\left(\frac{p}{1-p}\right)$$

**Step 2**

Per m=1,…,M:

Calcoliamo $r_{im} = -\left[\frac{\delta L(y_i, F(x_i))}{\delta F(x_i)}\right]_{F(x)=F_{m-1}(x)}$

il termine tra parentesi, come abbiamo già visto, è la derivata della Loss Function, che noi abbiamo già calcolato come:

$$-Osservazione * \log(odds) + \log\left(1 + e^{\log(odds)}\right)$$

$$\frac{\mathrm{d}}{d\log(odds)} = -Osservazione + \frac{e^{\log(odds)}}{1 + e^{\log(odds)}}$$

otteniamo quindi, considerando il segno negativo:

$$Osservazione - \frac{e^{\log(odds)}}{1 + e^{\log(odds)}}$$

che possiamo riscrivere come

$$Osservazione - p$$

quello che otteniamo è un "residuo" calcolato come la probailità delle osservazioni meno la probabilità della previsione.

**Step 3**

Modelliamo un albero per il "residuo" ottenuto e creiamo le regioni $R_{jm}$, dove $m$ indica l'abero che stiamo creando, il valore $j$ indica il nodo foglia ottenuto.
Per J=1,..., $J_m$ dobbiamo calcolare i valori di output:

$$\gamma_{im} = argmin_{\gamma} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$$

In questo caso il valore di output in ogni regione sarà :

$$\gamma = \frac{\sum_{x_i \in R_{ij}} r_{ij}}{\sum_{x_i \in R_{ij}} p_{ij} * (1 - p_{ij})}$$

**Step 4**

Come per i problemi di regressione, possiamo calcolare la predizione per i problemi di classificazione tramite Gradient Boosting con:

$$F_m(x) = F_{m-1}(x) + \theta \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$$

dove il primo termine sarà il valore $F_0(x) = \log(\frac{p}{1-p})$ e il secondo sarà il valore $\gamma$ risultante dall' albero precedente moltiplicato per il grado di apprendimento $\theta$.

### *3.7.8 XGBoost*

XGBoost è un algoritmo di Machine Learning basato su alberi decisionali che può essere considerato un estensione dell' algoritmo di Gradient Boosting, come suggerisce il nome XGBoost (eXtreme Gradient Boosting).
Presentato solo nel 2016, impostando in maniera ottimale i parametri si ottengono delle performance ottime.
I miglioramenti principali dell'algoritmo si possono racchiudere in questi 4 punti:

- Regolarizzazione: XGBoost penalizza modelli più complessi introducendo funzioni di costo basata su regression LASSO (Least Absolute Shrinkage and Selection Operator) e Regressione Ridge per prevenire overfitting
- Una migliore gestione delle informazioni mancanti nei dataset
- XGBoost utilizza un algoritmo migliorato per la ricerca dello split ottimale
- XGBoost utilizza un metodo built-in di Cross Validation, permettendo di ottimizzare il numero di iterazioni boosting da effettuare.

Per descrivere l'algoritmo XGBoost si può partire proprio dalle Loss Function ricavate con il Gradient Boosting, rispettivamente per i problemi di regressione e classificazione abbiamo:

$$\frac{1}{2}\ (Osservazione - Predizione)^2$$

$$-Osservazione * \log(odds) + \log\left(1 + e^{\log(odds)}\right)$$

XGBoost utilizza queste Loss Function per la costruzione di alberi minimizzando questa equazione:

$$[\sum_{i=1}^{n} L(y_i, p_i)] + \frac{1}{2}\lambda O_{value}^2$$

di cui il primo termine è la Loss Function sopra riportata a seconda che il problema sia di regressione o classificazione, il secondo termine è noto come termine di Regolarizzazione, l'obiettivo è trovare un Output Value ($O_{value}$) per un nodo foglia tale da minimizzare l'equazione.
Notiamo anche una somiglianza del secondo termine anche con la Ridge Regression, sia per la presenza del termine quadratico ma soprattutto per il fattore $\lambda$ , un termine di penalità definito come parametro di tuning.
Dal momento che l'algoritmo di XGBoost parte ad ottimizzare l'Output Value del primo albero, possiamo sostituire la predizione $p_i$ con la predizione iniziale aggiunto al valore dell'output del nuovo albero:

$$[\sum_{i=1}^{n} L(y_i, p_i^0 + O_{value})] + \frac{1}{2}\lambda O_{value}^2$$

Escludendo dall'equazione il secondo termine di regolarizzazione, otteniamo una funzione molto simile a quella vista per il Gradient Boosting, ma la tecnica XGBoost differisce da quest'ultima proprio per il calcolo dell' l'Output Value, che si traduce in un modo diverso di differenziare la Loss Function.
Mentre nel Gradient Boosting avevamo due distinte equazioni, la tecnica XGBoost utilizza la stessa metodologia sia per i problemi di classificazione che di regressione, precisamente utilizza un'approssimazione del polinomio di Taylor di secondo ordine:

$$L(y, p_i + O_{value}) \approx L(y, p_i) + \left[\frac{d}{dp_i} L(y, p_i)\right] O_{value} + \frac{1}{2}[\frac{d^2}{dp_i^2} L(y, p_i)] O_{value}^2$$

che è possibile scomporre in:

- $L(y, p_i)$ : la Loss Function della previsione precedente

- $\left[\frac{d}{dp_i} L(y, p_i)\right] O_{value}$: derivata prima della Loss Function, che indicheremo con $g$ in quanto connessa al concetto di gradiente

- $\frac{1}{2}[\frac{d^2}{dp_i^2} L(y, p_i)] O_{value}^2$ : derivata seconda della Loss Function, che indicheremo con $h$ in quanto connessa all' hessiana.

Ora riscriviamo la sommatoria $[\sum_{i=1}^{n} L(y_i, p_i^0 + O_{value})] + \frac{1}{2}\lambda O_{value}^2$ come:

$$L(y_1, p_1) + g_1 O_{value} + \frac{1}{2} h_1 O_{value}^2 +$$
$$+ L(y_2, p_2) + g_2 O_{value} + \frac{1}{2} h_2 O_{value}^2 + \cdots +$$
$$+ L(y_n, p_n) + g_n O_{value} + \frac{1}{2} h_n O_{value}^2 + \frac{1}{2}\lambda O_{value}^2$$

Dal momento che vogliamo calcolare il valore degli Output possiamo eliminare i termini dell'equazione dove non compare $O_{value}$ , ottenendo:

$$(g_1 + g_2 + \cdots + g_n) O_{value} + \frac{1}{2}(h_1 + h_2 + \cdots + h_n + \lambda) O_{value}^2$$

per trovare il valore minimo poniamo la derivata dell'equazione appena trovata $\frac{d}{dO_{value}} = 0$

risolvendo rispetto a $O_{value}$ ottengo :

$$O_{value} = \frac{-(g_1 + g_2 + \cdots + g_n)}{h_1 + h_2 + \cdots + h_n + \lambda}$$

• Se usiamo XGBoost per problemi di regressione:
Ricordando che la Loss Function per un problema di regressione è:

$$L(y_i, p_i) = \frac{1}{2}(y_i - p_i)^2$$

posso calcolare la derivata prima, il gradiente $g_i$:

$$g_{i=} \frac{d}{dp_i}\frac{1}{2}(y_i - p_i)^2 = -(y_i - p_i)$$

posso calcolare la derivata seconda, l' hessiana $h_i$:

$$h_i = \frac{d^2}{dp_i^2}\frac{1}{2}(y_i - p_i)^2 = \frac{d}{dp_i} - (y_i - p_i) = 1$$

ecco che possiamo riscrivere $O_{value}$ come:

$$O_{value} = \frac{Somma\ degli\ errori\ residui}{numero\ dei\ residui + \lambda}$$

• Se usiamo XGBoost per i problemi di classificazione:
Ricordando che Loss Function per un problema di classificazione è:

$$L(y_i, p_i) = -[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

per semplificare il calcolo della derivata esprimo l'equazione in funzione di *log(odds)* e calcolo la derivata prima e seconda come visto nei paragrafi precedenti:

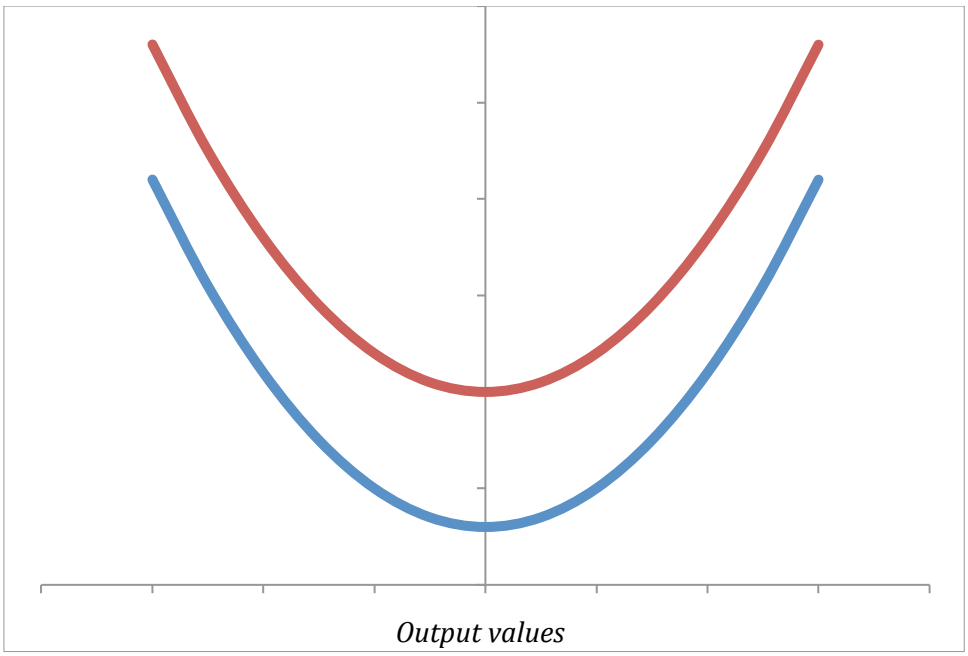
$$g_{i=}\frac{d}{d\ log(odds)}L(y_i,\log(odds)_i) = -y_i + \frac{e^{\log(odds)}}{(1+e^{\log(odds)})} = -(y_i - p_i)$$

$$h_i = \frac{d^2}{d\log(odds)^2}L(y_i,\log(odds)_i) = \frac{e^{\log(odds)}}{(1+e^{\log(odds)})} * \frac{1}{(1+e^{\log(odds)})}$$
$$= p_i * (1-p_i)$$

ora possiamo riscrivere $O_{value}$ come:

$$O_{value} = \frac{\sum Somma\ degli\ \ errori\ residui}{\sum\ [p_i * (1-p_i)] + \lambda}$$

SI ricordi che per ottenere il valore dell'Output Value, che si parli di regressione o classificazione, abbiamo omesso i termini $L(y,p_i)$ dal calcolo, questo a livello grafico si traduce in:




Ovvero le due parabole rappresentano $[\sum_{i=1}^{n} L(y_i, p_i^0 + O_{value})] + \frac{1}{2}\lambda O_{value}^2$ in rosso e $\sum g_i\, O_{value} + \frac{1}{2}\sum h_i\, O_{value}^2$ in blu, con stessa coordinata x per il valore minimo $O_{value}$.

Non solo abbiamo considerato l'equazione al netto di $L(y_i, p_i)$ perché non influenzava il calcolo dell'Output Value, ma anche perché l'equazione semplificata è quella che ci serve per il calcolo del *Similarity Score*.

Il *Similarity Score* è un indice che misura la qualità degli errori residui in ogni nodo foglia, viene calcolato come:

$$\frac{(Somma\ degli\ errori\ residui)^2}{Numero\ deli\ errori\ residui + \lambda}$$

Un *Similarity Score* basso indica che all'interno dello stesso nodo foglia abbiamo residui molto diversi tra di loro, che tendono quindi ad annullarsi a vicenda, al contrario, se sono molto simili (quindi con lo stesso segno), il numeratore aumenta e si ottiene un *Similarity Score* alto.
Non solo ci da una prima indicazione sugli errori residui in ogni nodo foglia ma serve anche a calcolare il *gain*, una condizione di split nell'algoritmo di XGBoost.
Essendo una condizione di split, si prendono in considerazione il nodo padre e i due nodi foglia che ne derivano, e si calcola come:

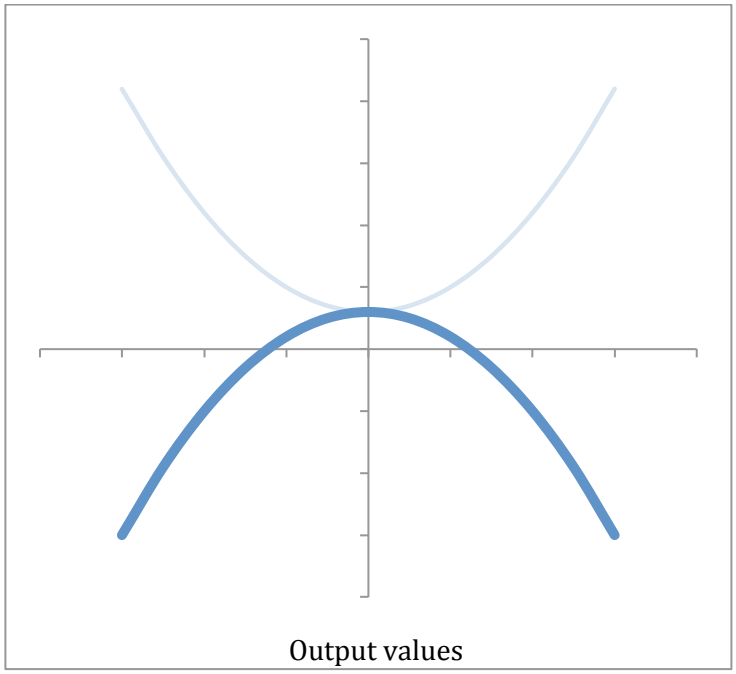
$$Gain = Similarity_{nodo\ foglia\ 1} + Similarity_{nodo\ foglia\ 2} - Similarity_{nodo\ foglia\ padre}$$

La condizione di split che genera il *gain* più alto viene scelta.

Per il calcolo del *Similarity Score* XGBoost moltiplica tutto per (-1), ottenendo:

$$-\sum g_i\, O_{value} - \frac{1}{2}\sum h_i\, O_{value}^2$$

questo a livello grafico si traduce con rotazione della parabola oltre la retta $y = 0$




la coordinata $y$ individuata dal punto più alto della parabola è proprio il *Similarity Score.*
Nelle applicazioni dell'algoritmo di XGBoost può variare e solitamente il Similarity Score che si ottiene è il doppio del valore della coordinata $y$ trovata, questo perché:

sostituendo i valori $O_{value}$ trovati all'interno dell'equazione:

$$-(g_1 + g_2 + \cdots + g_n)O_{value} - \frac{1}{2}(h_1 + h_2 + \cdots + h_n + \lambda)O_{value}^2$$

trovo:

$$-(g_1 + g_2 + \cdots + g_n)\frac{-(g_1 + g_2 + \cdots + g_n)}{h_1 + h_2 + \cdots + h_n + \lambda} -$$

$$-\frac{1}{2}(h_1 + h_2 + \cdots + h_n + \lambda)(\frac{-(g_1 + g_2 + \cdots + g_n)}{h_1 + h_2 + \cdots + h_n + \lambda})^2$$

e svolgendo i calcoli si ottiene il si ottiene la definizione del *Similarity Score* :

$$Similarity\ Score = \frac{1}{2}\frac{(g_1 + g_2 + \cdots + g_n)^2}{h_1 + h_2 + \cdots + h_n + \lambda}$$

che si differenzia dalla definizione data in precedenza per il fattore moltiplicativo $\frac{1}{2}$, nelle applicazioni XGBoost viene omesso, non influenza le decisioni finali, in quanto il *Similarity Score* è una misura relativa e i risultati saranno confrontabili allo stesso modo.

Ora se stiamo usando XGBoost per un problema di regressione , e come Loss Function stiamo utilizzando:

$$L(y_i, p_i) = \frac{1}{2}(y_i - p_i)^2$$

Sostituiamo $g_i$ con la derivata prima della Loss Function, che sarebbe il residuo negativo, ecco quindi che al numeratore otteniamo la somma dei residui al quadrato.
Sostituiamo anche $h_i$ con la derivata seconda della Loss Function,che abbiamo già visto corrispondere a "1", ottenendo quindi il *Similarity Score* per la regressione:

$$Similarity\ Score = \frac{1}{2}\frac{(somma\ degli\ errori\ residui)^2}{numero\ dei\ residui + \lambda}$$

se stiamo usando XGBoost per un problema di regressione , sostituiamo la derivata prima e seconda della Loss Function per Classificazione, come avevamo già trovato:

$$g_{i=}\frac{d}{d\ log(odds)}L(y_i, \log(odds)_i) == -(y_i - p_i)$$

$$h_i = \frac{d^2}{d \log (odds)^2} L(y_i, \log(odds)_i) == p_i * (1 - p_i)$$

e otteniamo il *Similarity Score* per i problemi si Classificazione

$$Similarity\ Score\ \frac{(Somma\ degli\ errori\ residui)^2}{\sum\ [p_i * (1 - p_i)] + \lambda}$$

# 4. Machine Learning per il Credit Scoring

## *4.1 Introduzione*

Uno degli ambiti di cui oggi si sente parlare spesso è quello del Machine Learning e Apprendimento Automatico.
Tramite il Machine Learning si insegna a computer e macchine a compiere azioni e attività imparando dall'esperienza. Attraverso modelli matematici e algoritmi i sistemi vanno a migliorare le proprie prestazioni in modo "adattivo", si parla di apprendimento automatico, in inglese *ensamble learning*.
Il termine Machine Learning è stato coniato nel 1959 per merito di Arthur Lee Samuel, uno scienziato americano pioniere nel settore dell'Intelligenza Artificiale. Da una prospettiva informatica, la vera rivoluzione è stata che, anziché dover progettare un codice di programmazione con cui, passo dopo passo, si indica alla macchina che cosa deve fare, con questa tecnologia vengono forniti al computer solo dei set di dati inseriti in un generico algoritmo, che è in grado di sviluppare una propria logica per svolgere la funzione o le attività richieste.

Insieme al Machine Learning anche termini come Intelligenza artificiale (AI) e Deep Learning(DL) hanno acquisito una notevole rilevanza, ma per quanto facciano parte dello stesso mondo non sono sinonimi.
L'Ai è un campo dell' informatica e si interessa di creare sistemi che svolgono attività che normalmente richiederebbero l' intelligenza umana, invece dando come input delle regole e algoritmi prestabiliti le macchine riescono a completare le attività.
"Intelligenza artificiale" quindi è un termine generico che comprende ML e DL.
Il Machine Learning di concentra su quegli approcci che permettono alle macchine di imparare autonomamente dai dati in input, mentre il Deep Learning è un sottogruppo del ML e incorpora modelli e algoritmi computazionali che imitano l'architettura delle reti neurali biologiche nel cervello.

Il ML unisce la statistica per quanto riguarda la relazione e l'analisi dei dati e l'informatica per la costruzione di modelli da dataset molto ampi.
Esistono diversi tipi di apprendimento, tra i più noti il *supervised learning* e *unsupervised learning.*

Nel supervised learning forniamo alla macchina un dataset iniziale, il training set, composto dall'osservazione e il risultato atteso. Quello che fa la macchina è proprio cercare di trovare la relazione che intercorre tra questi dati, in modo da poter poi applicare tale relazione alle osservazioni future e generare variabili di risposta. Esempi di algoritmi con supervised learning sono proprio gli alberi decisionali, Random Forest e Boosting.
Di contro nel unsupervised learning le etichette dei dati di input non sono note, non sappiamo a priori su quali dati calcolare la relazione, motivo per cui questi metodi studiano solo i pattern delle caratteristiche dei dati di input.
Gli obiettivi dell'unsupervised learning sono quindi: capire la distribuzione dei dati, il raggruppamento dei dati sulla base di caratteristiche simili o la riduzione delle loro dimensioni, al fine di fornirne una versione più sintetica ed efficace.
Questo approccio è frequentemente utilizzato per pre-processare i dati prima di fornirli ad altri algoritmi di supervised learning come dati di input.

## *4.2 Preanalisi dei dati*

La crescita esponenziale degli strumenti finanziari e del mercato del credito ha reso necessario sviluppare modelli di credit scoring per la valutazione del rischio. Ecco che un modello di credit scoring basato su alberi decisionali, può essere d'aiuto per classificare il buon credito e il cattivo credito.
I modelli sviluppati nelle prossime pagine sarano infatti basati su alberi decisionali con l'utilizzo di algoritmi avanzati quali Random Forest e Boosting realizzati con il linguaggio di programmazione R.

Si è scelto di applicare queste tecniche di Machine Learning a un settore specifico: quello manufatturiero elettronico, precisamente a tutte le aziende appartenenti al codice ATECO C26, che rientrano nelle seguenti tipologie:

26.1 FABBRICAZIONE DI COMPONENTI ELETTRONICI E SCHEDE ELETTRONICHE
26.2 FABBRICAZIONE DI COMPUTER E UNITÀ PERIFERICHE
26.3 FABBRICAZIONE DI APPARECCHIATURE PER LE TELECOMUNICAZIONI
26.4 FABBRICAZIONE DI PRODOTTI DI ELETTRONICA DI CONSUMO AUDIO E VIDEO
26.5 FABBRICAZIONE DI STRUMENTI E APPARECCHI DI MISURAZIONE, PROVA E NAVIGAZIONE; OROLOGI
26.6 FABBRICAZIONE DI STRUMENTI PER IRRADIAZIONE, APPARECCHIATURE ELETTROMEDICALI ED ELETTROTERAPEUTICHE
26.7 FABBRICAZIONE DI STRUMENTI OTTICI E ATTREZZATURE FOTOGRAFICHE
26.8 FABBRICAZIONE DI SUPPORTI MAGNETICI ED OTTICI

I dati delle aziende su cui lavorare sono stati estratti dal database AIDA (Analisi Informatizzata delle Aziende Italiane), per un totale di 10275 osservazioni (bilanci) per 1502 aziende; i dati estratti sono basati sullo storico di 10 anni a partire dal 2010, in media abbiamo quindi 7 bilanci annuali disponibili per azienda.

Nei bilanci annuali sono riportate per ogni azienda le informazioni relative ad:
- Informazioni anagrafiche
- Informazioni commerciali e legali
- Dati finanziari
- Indici finanzari

Dalle informazioni anagrafiche in particolare è possibile risalire agli eventi che le singole aziende hanno subito in passato, eventi come cessazione, scioglimento, fusione ecc, che ci danno un'indicazione sul tipo di azienda e su come considerarla in ottica della realizzazione del modello.
Ecco quindi che in base allo Stato Giuridico di ciascun bilancio per ogni azienda, possiamo assegnare un flag :

- Flag 0: società sana senza particolari segnalazioni

- Flag 1: società sana in condizioni particolari, quali:
-
  - Concordato preventivo
  - Fallimento
  - Amministrazione giudiziaria
  - Accordo di ristrutturazione dei debiti
  - Chiusura del fallimento
  - Altre cause
  - Liquidazione giudiziaria
  - Motivo non precisato
  - 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, quali:

  - 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 di 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, quali:
  - Fusione mediante incorporazione in altra società
  - Scissione
  - Trasferimento sede all' estero
  - Fusione mediante costituzione di nuova società
  - Cessione azienda
- Flag 3: società sana in condizioni particolari, quali:
  - cessata
  - cancellata dal registro impresa
  - trasferimento in altra provincia
  - cancellata d'ufficio a seguito istituzione cciaa di fermo, di monza,..

I flag che utilizzeremo in particolare per l'analisi saranno Flag 0, Flag 1 e Flag 2, considereremo società sane e società per cui si sono manifestati eventi legati a default e liquidazione.

Una volta scaricati i dati da Aida e assegnati i Flag alle varie società è necessario preparare i dati per la costruzione del modello. Come primo passo si "pulisce" il dataset da quelle osservazioni che presentano errori dovuti magari all' esportazione dal DataBase, questo è uno step necessario perché volendo realizzare un modello su R i dati devono essere formalizzati, e devono presentare le giuste caratteristiche.
Considereremo gli indicatori come variabili su cui costruire il modello e che determineranno i criteri di split, precisamente saranno:

- Indicatori Ordinari: quali ad esempio variazione pecentale ricavi o variazione percentuale EBITDA…

- Indicatori di Redditività: come EBIT/Ricavi, ROA, ROE…

- Indicatori di produttività e struttura operativa: Consumi/Costi Operativi, Costo del lavoro/Costi Operativi…

- Indicatori di liquidità e struttura finanziaria: Patrimonio Netto/Debiti Totali, liquidità su passivo, liquidità su attivo…

Su questi valori andiamo ad eseguire un ulteriore analisi, essi vengono limitati all'interno dell'intervallo tra il quinto e il novantacinquesimo percentile, questo per eliminare valori errati, quindi eccessivamente alti o bassi e vari outliers.

Un'operazione fondamentale da effettuare è quella dell'analisi della correlazione, sia tra le variabili per evidenziare quelle che hanno un alto grado di correlazione ma soprattuto per calcolare quella tra il Flag e la variabile $x_i$, questo perché avendo assegnato un Flag per ogni osservazione, mi aspetto che il segno della correlazione tra la variabile Indicatore e l' individuzione di un evento di default o liquidazione sia corretta.
In particolare il segno deve rispettare il significato dell'indicatore, quindi se maggiore è il valore dell'indicatore e minore è la probabilità di default mi aspetto una correlazione negativa, viceversa se maggiore è il valore e maggiore è la probabilità di default mi aspetto una correlazione positiva.
Dall'analisi dei risultati non tutti gli indicatori presentavano il segno corretto, motivo per cui non verranno inseriti nei dataset utilizzati per realizzare il modello.
Insieme agli indicatori ricavati dalla preanalisi sono stati aggiunti delle variabili *dummy* per fornire due caratteristiche descrittive in più: sulla posizione e sulla forma giuridica d'impresa.

Nell'applicazione degli algoritmi di Machine Learning si sono utilizzati due dataset diversi come input, nel primo il flag 1 è segnato per tutte le osservazioni della società che presentano almeno un evento default in uno dei bilanci, nel secondo si assegna il flag 1 solo all'anno in cui si verifica l'evento default per la società.
Quello che vogliamo è costruire modelli che siano consistenti per entrambe le casistiche, tramite l'utilizzo di tecniche di Machine Learning con linguaggio di programmazione R.

## *4.3 Applicazione algoritmi di Machine Learning in R*

R è un linguaggio di programmazione specifico per l'analisi statisica dei dati. La prima versione risale al 1993 e venne inizialmente scritto dal matematico Robert Gentleman e dallo statistico Ross Ihaka.
Il punto forte di R è la sua flessibilità nell'analisi e la rappresentazione dei dati, motivo per cui viene utilizzato anche per la realizzazione di modelli di Machine Learning.

Molti degli algoritmi richiesti non sono inclusi nell'ambito dell'installazione base di R. Tuttavia, è possibile utilizzare una raccolta di funzioni a seconda dello scopo, chiamata *package*.
I *packages* utilizzati per l'analisi dati svolta sono stati:
- *Tree e RPart* per la creazione di alberi decisionali
- *Random Forest* per l'applicazione dell'omonimo algoritmo
- *Gbm* per il Gradient Boosting
- *Xgboost* per l'algoritmo XGBoost

Verrano effettuate due analisi, a seconda del dataset iniziale su cui lavorare.
Nella prima analisi si sono considerate tutte le osservazioni, corrispondenti a tutti i bilanci esportati da Aida, ed è stato assegnato Flag 1 (a segnalare l'evento default) per tutte le osservazioni delle società per cui in almeno un anno si è verificata una delle situazioni sopra riportate per i Flag 1 e 2.
Nella seconda analisi si sono considerate le osservazioni delle società che hanno presentato l'evento default, ed è stato assegnato Flag 1 ai singoli anni delle società dove si è verificata una delle situazioni sopra riportate per i Flag 1 e 2.

## 4.4 Analisi 1

Il dataset utilizzato per quest'analisi è il risultato della preanalisi descritta nelle pagine precedenti, costituito da un insieme di indicatori che costituiranno i criteri di split. Per semplicità di lettura, in fase di output grafici, si è assegnato un codice ad ogni indicatore:

| | |
|---|---|
| SPA | I1 |
| SRL | I2 |
| SAS | I3 |
| SNC | I4 |
| Consorzio | I5 |
| SCARL | I6 |
| Nord | I7 |
| Sud/Isole | I8 |
| EBITDA/Ricavi | I9 |
| EBIT/ricavi | I10 |

| | |
|---|---|
| Utile corrente/ricavi | I11 |
| Risultato netto rettif/ricavi | I12 |
| Risultato netto rettif/AN | I13 |
| EBITDA/AN | I14 |
| ROA | I15 |
| O-ROI | I16 |
| ROE | I17 |
| ROE ante imposte | I18 |
| Val Agg Oper/Ricavi | I19 |
| Val Agg/ITN | I20 |
| Ricavi/AN | I21 |
| AC-mag/PC | I22 |
| Liq/PC | I23 |
| Liq/AN | I24 |
| Cap Circ/AN | I25 |
| Patr netto/AN | I26 |
| Riserve+utile/AN | I27 |
| Patr netto tang/AN | I28 |
| Patr netto/debiti totali | I29 |
| Patr netto tan/Debiti tot+PN | I30 |
| Patr netto tan/Debiti tot-Liq+PN | I31 |
| O-Patr.netto/deb fin-liq+PN | I32 |
| OF/RIC | I33 |
| OFN/RIC | I34 |
| OFN/EBITDA | I35 |
| OFN/EBIT | I36 |
| OFN/AN | I37 |
| OFN/Autof lordo | I38 |
| Autof Lordo/AN (=cash flow/attivo) | I39 |
| Deb totali/VA | I40 |
| Deb totali/Ric | I41 |
| PC/ric | I42 |
| Deb finanziari (stimati)/VA | I43 |
| Deb finanziari (stimati)/Ric | I44 |
| Debiti totali/EBITDA | I45 |
| Debiti finanziari/EBITDA | I46 |
| Ebitda-servizio debito/AN | I47 |

Oltre i 47 indicatori è stata aggiunta la variabile indipendente FLAG, una variabile binaria che sarà la variabile target, quella da prevedere, che identificherà l'evento default.
L'analisi verterà sull'applicazione dei metodi di machine learning descritti, per misurarne poi l'accuratezza, cominciamo richiamando la libreria (tree) per utilizzare le funzioni che ci serviranno per realizzare il primo modello di albero decisionale, partendo dall'insieme di Indicatori sopra riportato chiamato appunto "Indicatori":

```
library(tree)
data <- Indicatori
str(data)
```

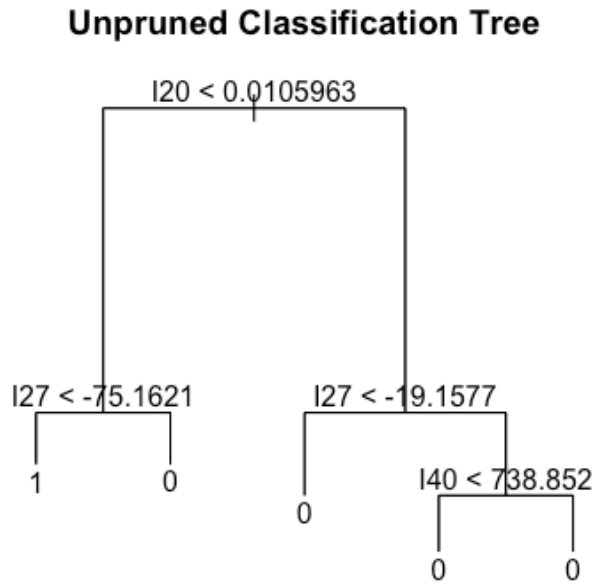
si assegna alla varabile FLAG una variabile di tipo fattore, può assumere solo valori binari, saranno i valori su cui misureremo l'accuratezza del modello, in base alla probabilità di misclassificazione, considerando valore "1" come evento default e valore "0" per società sana. Si procede poi all'utilizzo della funzione *tree* per realizzare il primo modello ad albero e la funzione plot per rappresentarlo

```
data$FLAG= as.factor(ifelse(data$FLAG==0,"0","1"))
str(data)
tree.data=tree(FLAG~.,data=data)
summary(tree.data)
plot(tree.data)
text(tree.data, pretty =0)
title(main=" Unpruned Classification Tree ")
tree.data
```

L'output generato è il seguente, 5 nodi finali, si noti la presenza dei 3 indicatori che meglio hanno soddisfatto le condizioni dell'Indice di Gini e i primi elementi statistici per la valutazione dell'albero come l'errore standard residuo medio, notiamo infatti come il rapporto sia calcolato su 10270 elementi (10275 totali meno i 5 nodi finali) e l'errore di misclassificazione.

```
Classification tree:
tree(formula = FLAG ~ ., data = data)
Variables actually used in tree construction:
[1] "I20" "I27" "I40"
Number of terminal nodes:  5
```

Residual mean deviance: 0.662 = 6799 / 10270
Misclassification error rate: 0.1277 = 1312 / 10275




Questo è un primissimo modello di output ad albero, ma non ci da molte indicazioni sulla sua accuratezza, sarà poi necessario verificare se effettuare il "pruning" dell'albero, diminuire i numeri di nodi finali per diminuire la varianza.
Per cominciare ad avere una prima misura della performance di previsione di un modello ad albero decisionale semplice dividiamo il dataset iniziale in Training data e Testing data(in rapporto 3:2), come è solito fare per gli algoritmi di supervised learning. La macchina studia le relazioni tra i dati del dataset fornito, nel nostro caso tra FLAG (che è la variabile di previsione) e gli indicatori e applicherà la funzione di previsione ai testing data e potremo avere una prima misura dell'accuratezza di previsione.

```
dim(data)
set.seed(2)
data.idx= sample(1:nrow(data),6000)
data.train=data[data.idx,]
data.test=data[-data.idx,]
dim(data.train)
```

```
dim(data.test)
tree.train=tree(FLAG~.,data=data.train)
summary(tree.train)
plot(tree.train)
text(tree.train)
title(main="Unpruned Train Tree")
data.train.pred=predict(tree.data, data.train, type="class")
data.test.pred=predict(tree.data,data.test, type="class")
table(predicted=data.train.pred, actual=data.train$FLAG)
table(predicted=data.test.pred,actual=data.test$FLAG)
accuracy=function(actual,predicted) {mean(actual == predicted)}
accuracy(predicted=data.train.pred, actual=data.train$FLAG)
accuracy(predicted=data.test.pred,actual=data.test$FLAG)
tm.accuracy <- accuracy(predicted=data.test.pred,actual=data.test$FLAG)
```

L'output mostra la probabilità di misclassificazione rispettivamente per il Training set e il Testing set, ottenendo un' accuratezza nella predizione dell' 0.8580702

*Training data*

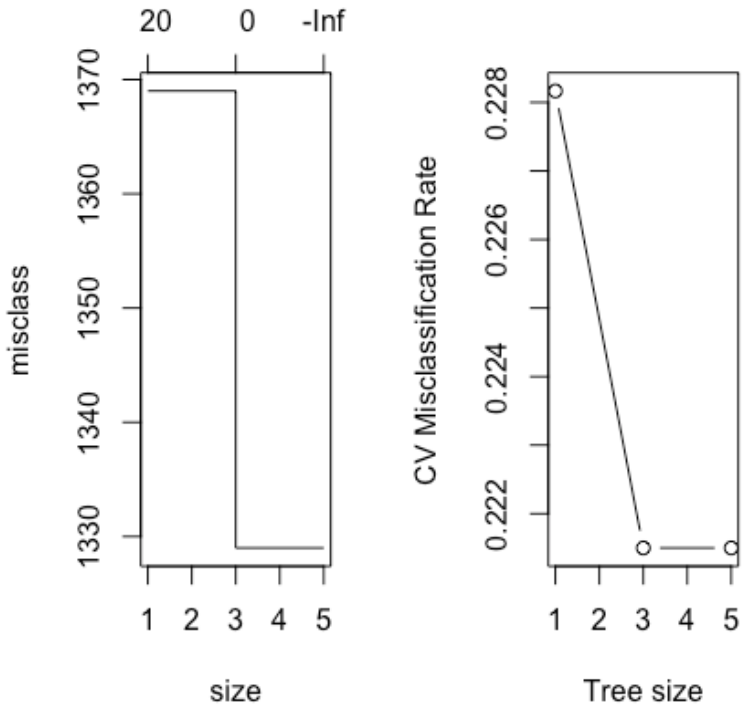
| | | actual | |
|---|---|---|---|
| | | 0 | 1 |
| predicted | 0 | 5093 | 631 |
| | 1 | 117 | 159 |

*Testing data*

| | | actual | |
|---|---|---|---|
| | | 0 | 1 |
| predicted | 0 | 3617 | 467 |
| | 1 | 94 | 97 |

I risultati possono essere migliorati, un primo accorgimento da fare è applicare il pruning dell'albero, e tramite output grafico visualizzeremo se è necessario diminuire i nodi finali e come l'errore di classificazione si comporta in base alla dimensione dell'albero.

*set.seed(3)*
*tree.data.cv=cv.tree(tree.data, FUN=prune.misclass)*
*min.idx=which.min(tree.data.cv$dev)*
*min.idx*
*tree.data.cv$size[min.idx]*
*tree.data.cv$dev / length(data.idx)*
*par(mfrow=c(1,2))*
*plot(tree.data.cv)*
*plot(tree.data.cv$size, tree.data.cv$dev / nrow(data.train), type = "b", xlab="Tree size",ylab = "CV Misclassification Rate")*




Come si evince dai grafici, l'errore di classificazione diminuisce all'aumentare del numero di nodi finali, non è necessario quindi "potare" l'albero per ottenere un errore di misclassificazione minore.

Il modello ad albero decisionale può essere migliorato, le varie tecniche di boostrap e bagging possono aumentare l'accuratezza del modello, di seguito il

confronto con un modello di regressione logistica, di bagging, di Random Forest e di Boosting.
L'accuratezza dei vari modelli sarà sempre calcolata mettendo a confronto i risultati ottenuti e quelli osservati, verificando quindi quanto il modello sia riuscito bene a prevedere i risultati, con la seguente funzione:

```
#prediction accuracy
calc_acc = function(actual, predicted) {mean(actual == predicted)}
```

Il modello di regressione logistica su R è descritto dalla funzione "glm", anche in questo caso si dividerà il dataset iniziale per "modellare" i dati del training set e misurare poi l'accuratezza con i testing data.

```
#logistic regression
library(pROC)
lg.data=glm(FLAG~., data=data.train, family = "binomial")
lg.data.pred=ifelse(predict(lg.data,data.test,"response")>0.5,1,0)
table(predicted=lg.data.pred, actual=data.test$FLAG)
lg.data.test.acc= calc_acc(predicted = lg.data.pred, actual=data.test$FLAG)
lg.data.test.acc
```

Il risultato ottenuto addestrando il dataset "Indicatori" tramite algoritmo di Regressione Logistica è 0.8692398.

Il risultato mostra un miglioramento rispetto all'albero decisionale, ma l'obiettivo è cercare di ottenere con consistenza delle previsioni accurate andando a realizzare degli algoritmi decision tree based.

## Bagging

Cominciamo dai modelli di Bagging e Random Forest, l' analisi sarà svolta insieme poiché da un punto di vista teorico i modelli molto simili, il Random Forest infatti è un caso particolare di Bagging, dove si utilizzano solo una parte delle variabili messe a disposizione.
Il fatto che Il Bagging utilizzi tutte le variabili non deve essere visto necessariamente come un vantaggio: richiede più calcolo computazionale e i risultati ottenuti rischiano di essere meno consistenti e più variabili.

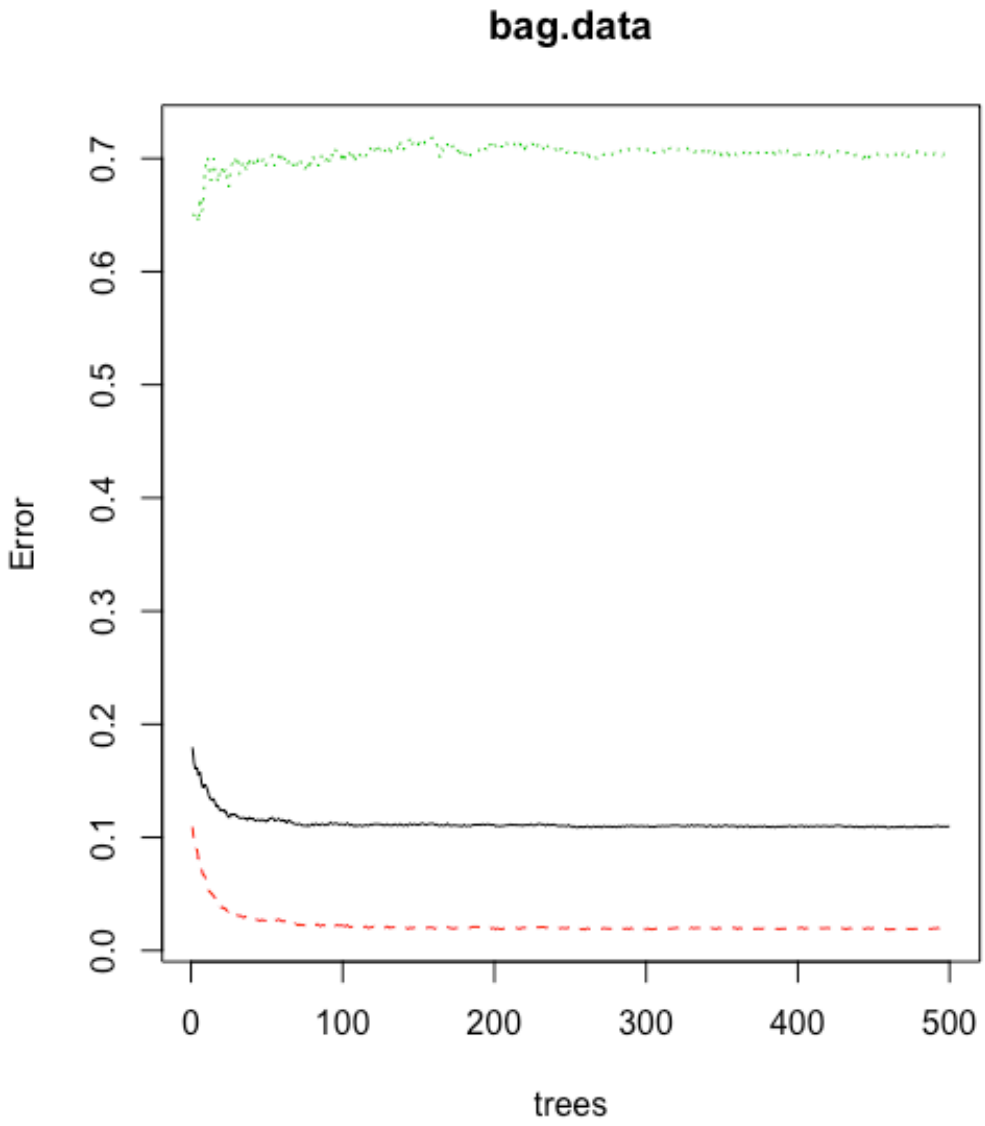
Entrambi gli algoritmi verranno eseguiti con la funzione R "*randomForest*"
Cominciando con quello di Bagging:

```
#bagging
library(randomForest)
bag.data=randomForest(FLAG~.,data=data.train,  mtry=47,importance=TRUE,
ntrees=500)
plot(bag.data)
bag.data
bag.data.pred=predict(bag.data,newdata=data.test)
table(predicted=bag.data.pred,actual=data.test$FLAG)
bag.data.test.acc= calc_acc(predicted =bag.data.pred, actual =data.test$FLAG)
bag.data.test.acc
```

Notiamo sempre la presenza del doppio set, uno per il training dei dati (bag.data.pred) e uno per testare le capacità previsionali del modello (bad.data.test).
L'output ottenuto da questo codice è il seguente:

```
Type of random forest: classification
                      Number of trees: 500
No. of variables tried at each split: 47

        OOB estimate of  error rate: 10.97%
```

E' un riassunto del modello di bagging creato, dove vengono riportate le caratteristiche principali, quindi il numero di alberi creati, il numero di variabili utilizzate nel modello(tutte le 47 variabili del dataset) e il tipo di albero generato (nel nostro caso albero di classificazione).
Si noti la presenza di una nuova voce: "OOB estimate of error rate", ci aspettiamo una stima degli OOB nel modello di bagging, perché sono gli errori generati dalla tecnica di Bootstrap, su cui i modelli di bagging si basano (come esposto nel capitolo sul bagging).
In questo caso "OOB estimate of error rate" sta a indicare proprio l'errore generato dal modello di bagging man mano che aggiungiamo alberi.
Con la funzione plot nel codice otteniamo anche un output grafico:




Un grafico che ci è familiare, mostra infatti come l'errore tende a diminuire e stabilizzarsi man mano che il numero di alberi aumenta, la curva che ci interessa è quella di colore nero, rappresentante l'errore del modello di bagging creato.
La funzione di accuracy del modello di bagging ottenuto restituisce un valore pari a 0.8866871.

## Random Forest

Il modello di Random Forest condivide l'impostazione del precedente modello di Bagging ma ha una caratteristica che lo contraddistingue: il parametro *mtry* .

Questo parametro indica il numero di variabili da analizzare ad ogni split dell'albero, mentre nel Bagging si utilizzano tutte le variabili, qui si imposta un valore inferiore, che indubbiamente influenza il valore di *accuracy.*
Il valore di default per mtry è solitamente pari alla radice delle variabili utilizzate nel dataset, ma in questo caso utilizzeremo una funzione per determinarlo:

```
#ottimizzazione paratmetro mtry
oob = trainControl(method = "oob")
cv_5 = trainControl(method = "cv", number = 5)

dim(data.train)
rf.grid=expand.grid(mtry=1:15)
set.seed(825)
data.rf.tune=train(FLAG ~ ., data = data.train,
                   method = "rf",
                   trControl = oob,
                   verbose = FALSE,
                   tuneGrid = rf.grid)
data.rf.tune
```

Con cui generiamo questo output:

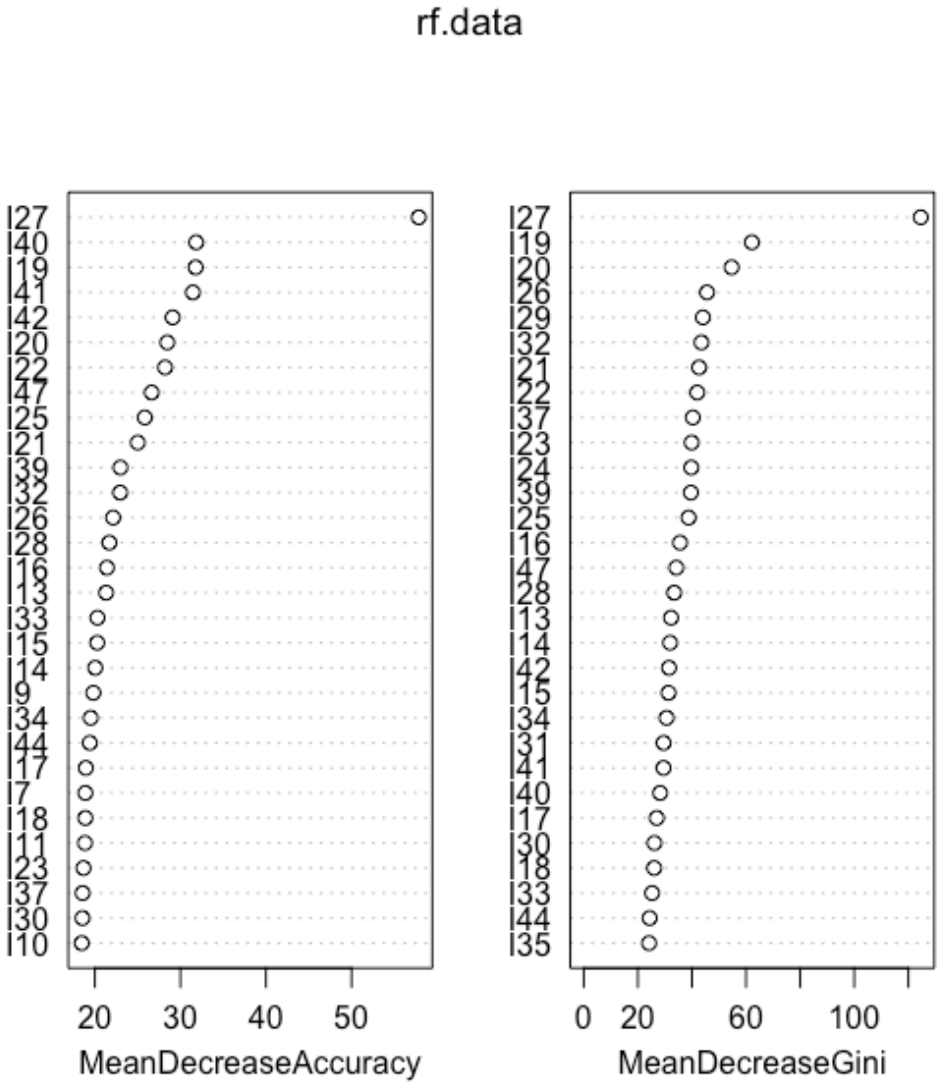
| | mtry | Accuracy | Kappa |
|---|---|---|---|
| 1 | | 0.8771667 | 0.1430066 |
| 2 | | 0.8850000 | 0.2786099 |
| 3 | | 0.8871667 | 0.3087064 |
| 4 | | 0.8886667 | 0.3275078 |
| 5 | | 0.8906667 | 0.3395885 |
| 6 | | 0.8883333 | 0.3284824 |
| 7 | | 0.8905000 | 0.3468168 |
| 8 | | 0.8883333 | 0.3314441 |
| 9 | | 0.8881667 | 0.3387099 |
| 10 | | 0.8896667 | 0.3499394 |
| 11 | | 0.8890000 | 0.3431607 |
| 12 | | 0.8913333 | 0.3606840 |
| 13 | | 0.8903333 | 0.3538673 |
| 14 | | 0.8920000 | 0.3609131 |
| 15 | | 0.8893333 | 0.3479755 |

Cosa si evince da questo output? R ci restituisce un processo iterativo, ad ogni valore di mtry restituisce i valori di accuracy e Kappa. La statistica Kappa è acnh'essa una misura di accuracy, la differenza sta nel fatto che compara l'accuratezza osservata con l'accuratezza prevista, motivo per cui il suo valore sarà inferiore a quello di Accuracy.
Per entrambi notiamo una similitudine: ottengono il valore massimo per mtry = 14, che sarà proprio il valore che utilizzeremo per parametrizzare il modello di Random Forest.

```
rf.data=randomForest(FLAG~.,data=data.train,  mtry=14,importance=TRUE,
ntrees=100)
rf.data
plot(rf.data)
importance(rf.data)
varImpPlot(rf.data)
rf.data.pred=predict(rf.data, newdata=data.test)
table(predicted=rf.data.pred, actual=data.test$FLAG)
rf.data.test.acc=calc_acc(predicted = rf.data.pred, actual = data.test$FLAG)
rf.data.test.acc
```

Ci aspettiamo un leggero miglioramento visto che abbiamo inserito il numero di variabili ottimali da considerare ad ogni split (come si nota dalla funzione randomForest dove è specificato *mtry=14).*
L'accuratezza generata dal modello Random Forest risulta essere 0.8879193

Dal codice notiamo anche il comando plot *varImpPlot(rf.data),* dove varImpPlot è una funzione che riporta in un grafico le variabili più importanti del dataset, quelle che contribuiscono a migliorare l'accuratezza del modello:




Si noti come gli Indicatori sono ordinati in base alla "MeanDecreaseAccuracy" e "MeanDecreasGini"

La "MeanDecreaseAccuracy" indica quanta accuratezza il modello perderebbe se eliminassimo dal dataset l'indicatore $Ix_i$ , la "MeanDecreaseGini" indica quanto l'indicatore $Ix_i$ contribuisca all'omogeneità del nodo e del nodo foglia nel modello RandomForest.

Gli Indicatori più importanti risultano essere quindi

| | |
|---|---|
| I27 | Riserve+utile/AN |
| I20 | Val Agg/ITN |
| I40 | Deb totali/VA |
| I19 | Val Agg Oper/Ricavi |

Non è un caso che queste variabili siano anche le stesse utilizzate dal primo modello dell'albero decisionale semplice come variabili per i criteri di split, come si nota anche dall'output ad albero generato.

## Boosting

Il modello di Boosting lavora in maniera diversa dal Bagging, come abbiamo ampiamente visto nel corrispettivo paragrafo, ogni albero generato dipende da quello precedente, e realizzando *weak learner* in serie, cerca di annullare di volta in volta gli errori generati.
Per il boosting utilizzeremo due funzioni, una per il gradient boostin (gbm), una per XGBoost(xgboost).

Per il modello di Gradiente Boosting introduciamo i 3 parametri principali di questa funzione:

- N.trees= numero di alberi generati, un numero elevato permette di ottenere un errore residuo basso, ma può portare a problemi di overfitting
- interaction.depth= il numero di splits da generare per ogni albero creato
- shrinkage= un parametro che esprime il grado di apprendimento, è lo stesso parametro $\theta$ visto nel paragrafo introduttivo al boosting, un valore troppo basso richiede un numero elevato di alberi generati per avere delle buon performance.

Può essere paragonato al termine di regolarizzazione della Ridge Regression, che tende a penalizzare la complessità del modello generato.

Prima di scrivere il codice per il modello è possibile fare il cosiddetto *Tune Boosting*, ovvero andare ad ottimizzare i parametri partendo dal nostro dataset, un concetto simile a quello che abbiamo fatto con il parametro mtry per il RandomForest:

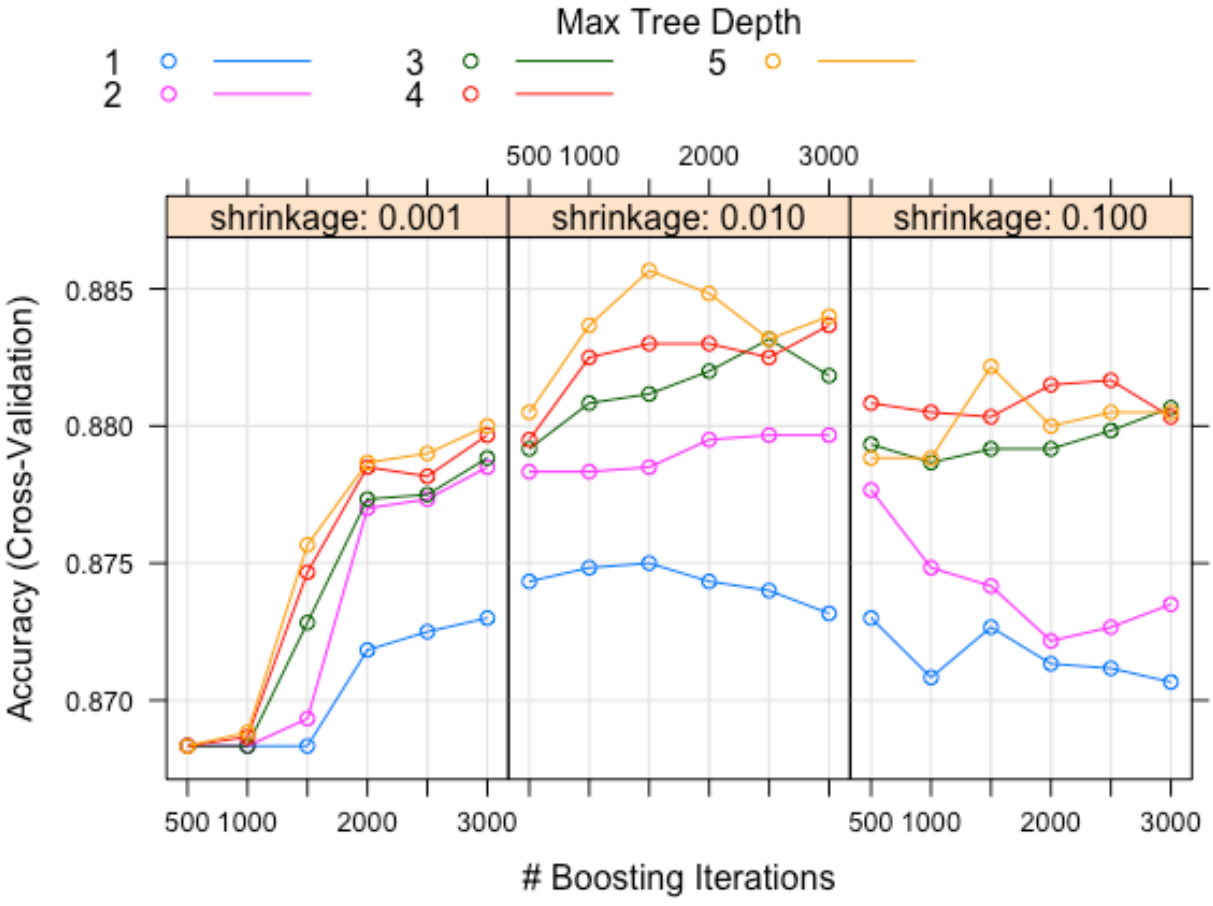
```
gbm.grid =  expand.grid(interaction.depth = 1:5,
                        n.trees = (1:6) * 500,
                        shrinkage = c(0.001, 0.01, 0.1),
                        n.minobsinnode = 10)

data.gbm.tune= train(FLAG ~ ., data = data.train,
```

*method = "gbm",*
*trControl = cv_5,*
*verbose = FALSE,*
*tuneGrid = gbm.grid)*

*plot(data.gbm.tune)*

L'output che si ottiene è un comodo grafico che ci mostra come l'*accuracy* vari a seconda dei parametri principali della funzione gbm.
Ecco che valuteremo per il nostro modello:

- n.trees > 3000
- interaction.depth = 5
- shrinkage = 0.01

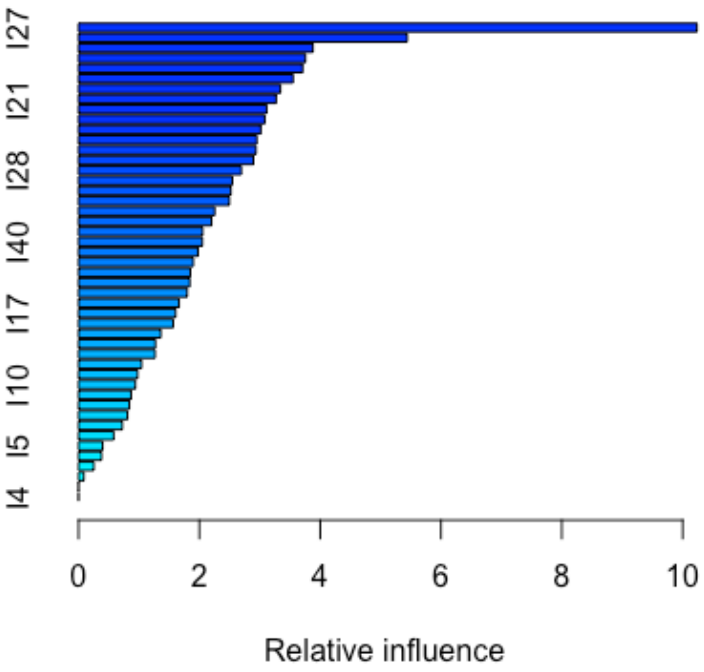


Possiamo ora scrivere il modello:

```
#Boosting
data$FLAG= as.numeric(ifelse(data$FLAG==0,"0","1"))
data.idx2= sample(1:nrow(data),6000)
data.train2=data[data.idx2,]
data.test2=data[-data.idx2,]
library(gbm)
set.seed(1)
data.boost=gbm(FLAG~., data=data.train2, distribution = "bernoulli", n.trees = 5000, interaction.depth =5, shrinkage =0.01)
plot(data.boost)
data.boost
summary(data.boost)
data.boost.test.pred=ifelse(predict(data.boost, newdata=data.test2, n.trees=5000, "response") >0.5, "1", "0")
table(predicted=data.boost.test.pred, actual= data.test2$FLAG)
boost.data.test.acc=calc_acc(predicted =data.boost.test.pred, actual = data.test2$FLAG )
boost.data.test.acc
```

Dall'output generato possiamo ricavare un grafico con relativa lista delle variabili (si riportano le 4 principali) e delle loro importanza relativa per la funzione di boosting *gbm:*

| var | rel.inf |
|---|---|
| I27 | 10.23713165 |
| I20 | 5.44024131 |
| I32 | 3.88033510 |
| I19 | 3.75597892 |




Dalla funzione di accuracy ricaviamo invece il valore dell'accuratezza del modello di Boosting pari a 0.8870256

## XGBoost

L'algoritmo XGboost rappresenta una delle migliori evoluzioni dei modelli di boosting,
La funzione utilizzata in R è *xgb* che presenta diversi parametri di controllo, tra i principali:

- **eta**: molto simile al parametro "shrinkage" della funzione *gbm,* indica il grado di apprendimento e controlla "quanta informazione" verrà utilizzata per il boosting dall'albero generato
- **max_depth**: controlla la profodità degli alberi generati (il numero dei nodi generati). Un numero alto indica alberi generati di dimensioni maggiori, questo non necessariamente è un vantaggio per almeno due motivi, alberi troppo profondi all'inizio avranno più importanza e prenderanno più informazioni dal dataset rispetto agli ultimi, in più il boosting si basa su creazione iterativa di piccoli *weak learners*, che per definizione non sono molto più grandi di uno *stump.*

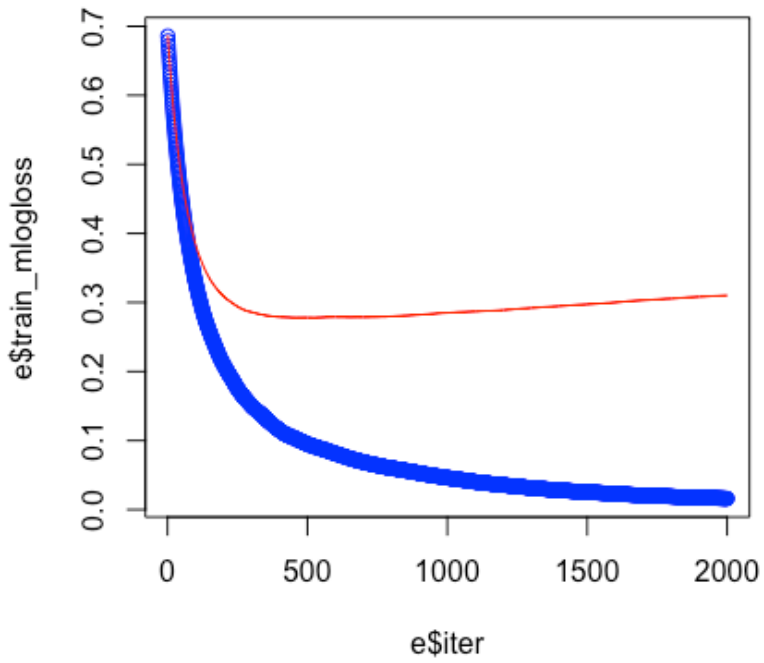
- **sub_sample**: determina se stiamo realizzando un modello di Boosting o un modello Boosting stocastico. Nel primo caso il parametro assume valore pari a "1", nel secondo un valore compreso tra "0" e "1" che indica la percentuale del dataset utilizzata per creare ogni albero. Il boosting stocastico viene usato quando si è in presenza di un dataset con molti outliers che potrebbero creare alta varianza.
- **gamma**= parametro di regolarizzazione, controlla la riduzione della loss function, un valore alto tende a interrompere in anticipo il numero di alberi generati, quindi per non aumentare la complessità dell'albero rinuncia a quegli alberi che riducono di poco l'errore generato.

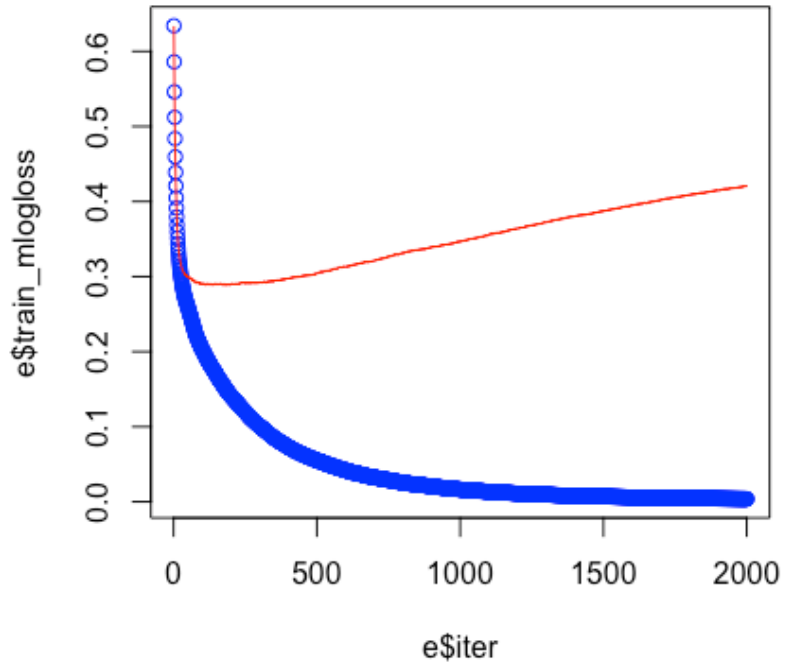
La scelta di questi valori influenza l'output generato e soprattuto per il parametro gamma si tende a scegliere il valore in maniere iterativa.
Una delle misure di valutazione utilizzate per valutare il modello di Boosting è la Log Loss, una funzione di costo, un valore basso indica una buona qualità del modello, ma è diverso dal concetto di Accuracy.
L'output per l'algoritmo di XGBoost, dopo avere sempre diviso il datates in training e testing data, è il seguente:

```
best_model <- xgb.train(params = xgb_params,
                        data = train_matrix,
                        nrounds = 2000,
                        watchlist = watchlist,
                        eta = 0.1,
                        max.depth =4,
                        gamma = 0,
                        subsample = 1,
                        colsample_bytree = 1,
                        missing = NA,
                        )
best_model
e <- data.frame(best_model$evaluation_log)
plot(e$iter, e$train_mlogloss, col = 'blue')
lines(e$iter, e$test_mlogloss, col = 'red')
```

Questa prima funzione restitutisce un output che dà una prima indicazione della qualità del modello, otteniamo infatti un grafico dove sono riportate le funzioni di costo per il training e il testing set, rispettivamente train_mlogloss (linea blu) e test_mlogloss (linea rossa).

L’obiettivo è ottenere curve che convergano il più possibile verso valori bassi e entrambe con andamento decrescente, e questo si ottiene andando a modificare i parametri descritti.
Una parametrizzazione non corretta può portare ad output grafici del tipo:



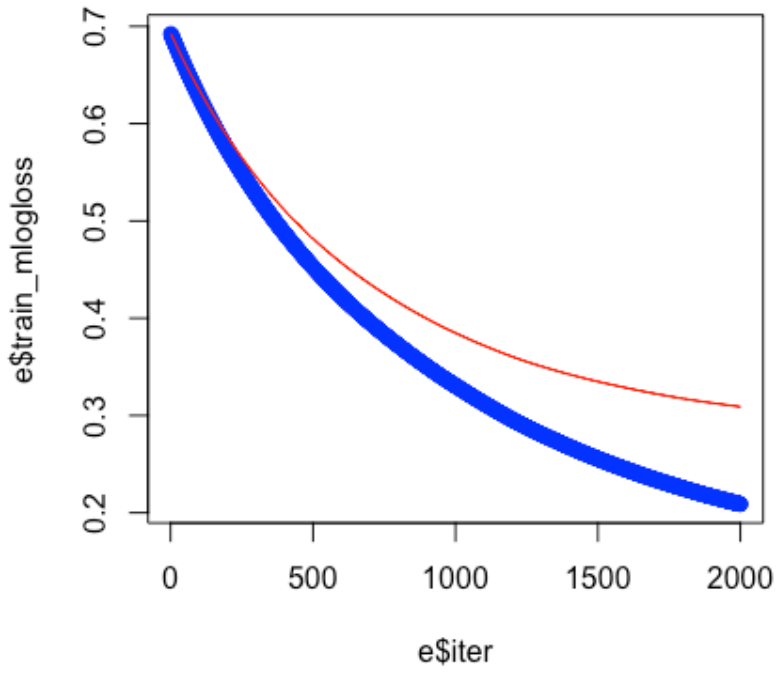



Questi output non sono soddisfacenti, non offrono una buona qualità del modello, vediamo che la curva di costo per il testing set non converge e in alcuni casi dopo un certo numero di iterazioni boosting tende addirittura ad aumentare, e sappiamo che ad un valore più alto della Log Loss corrisponde una scarsa efficacia del modello, ne consegue un accuracy non ottimale.
Ecco che la scelta dei parametri diventa fondamentale, con la funzione:

```
best_model <- xgb.train(params = xgb_params,
                        data = train_matrix,
                        nrounds = 2000,
                        watchlist = watchlist,
                        eta = 0.001,
                        max.depth =8,
                        gamma = 0,
                        subsample = 1,
                        colsample_bytree = 1,
                        missing = NA,
                        )
```

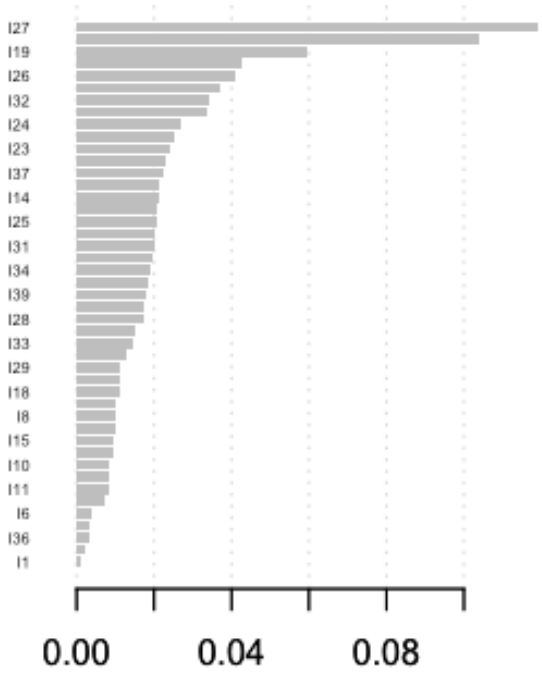
otteniamo un risultato decisamente migliore:

```
                                    evaluation_log:
iter train_mlogloss test_mlogloss
   1       0.692434      0.692504
   2       0.691718      0.691861
                                                ---
1999       0.208956      0.309047
2000       0.208875      0.309007
```




Le curve convergono e hanno un andamento simile, questa caratteristica è sinonimo di un risultato consistente, infatti ottenere un valore della funzione di costo bassa per il training set, non necessariamente si traduce con un valore basso per il testing set, possono crearsi problemi di overfitting , come nei 2 casi visti in precedenza.

Possiamo poi ricavare un'indicazione sulle variabili più importanti per il modello di XGBoost e calcolarne finalmente l'*accuracy:*




L'accuracy tramite XGBoost è la più alta misurata con gli algoritmi esposti, otteniamo infatti un valore pari a 0.9082244.

## 4.5 Analisi 2

La seconda analisi verterà sempre sulla realizzazione di algoritmi decision tree based tramite R, con l'utilizzo di un dataset iniziale diverso.
In questo caso viene assegnato il valore "1" (indicante l'evento default) solo al singolo anno dove si riportano gli eventi descritti dai FLAG 1 e 2 introdotti nel capitolo di preanalisi dei dati.
Le osservazioni sono limitate ai bilanci delle società per cui si verifica l'evento default e sono quindi inevitabilmente in numero inferiore, abbiamo un dataset composto da 1399 osservazioni/bilanci per 48 variabili/Indicatori.
L'analisi sui dati prima di lavorarli con R seguirà le stesse linee guida dell'analisi 1, ma considerando la diversità dei dataset ne consegue che gli indicatori utilizzati saranno in parte diversi, sempre come conseguenza del segno della correlazione FLAG/variabili che deve rispettare il significato dell'Indicatore.
Le variabili che comporrano il dataset sono:

| | |
|---|---|
| SPA | I1 |
| SRL | I2 |
| SAS | I3 |
| SNC | I4 |
| Consorzio | I5 |
| SCARL | I6 |
| Nord | I7 |
| Sud/Isole | I8 |
| EBITDA/Ricavi | I9 |
| EBIT/ricavi | I10 |
| Utile corrente/ricavi | I11 |
| Risultato netto rettif/ricavi | I12 |
| Risultato netto rettif/AN | I13 |
| EBITDA/AN | I14 |
| ROA | I15 |
| O-ROI | I16 |
| ROE | I17 |
| ROE ante imposte | I18 |
| Val Agg Oper/Ricavi | I19 |
| Val Agg/ITN | I20 |
| Ricavi/AN | I21 |
| O-Circ Oper (mag+cl-for)/Ric | I22 |
| AC-mag/PC | I23 |

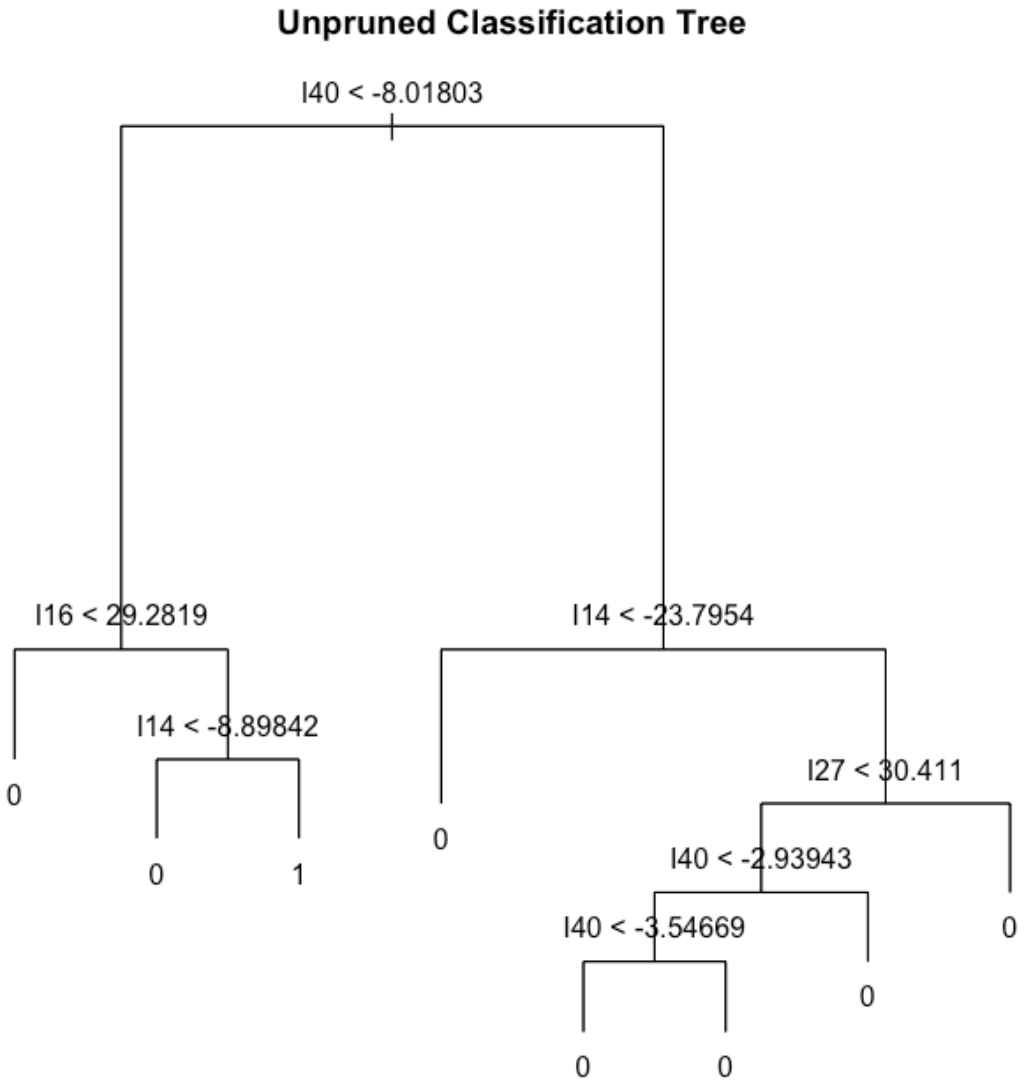
| | |
|---|---|
| Liq/PC | I24 |
| Cap Circ/AN | I25 |
| Patr netto/AN | I26 |
| Riserve+utile/AN | I27 |
| Patr netto tang/AN | I28 |
| Patr netto/debiti totali | I29 |
| Patr netto tan/Debiti tot+PN | I30 |
| Patr netto tan/Debiti tot-Liq+PN | I31 |
| O-Patr.netto/deb fin-liq+PN | I32 |
| OF/RIC | I33 |
| OFN/RIC | I34 |
| OFN/EBITDA | I35 |
| OFN/EBIT | I36 |
| OFN/AN | I37 |
| OFN/Autof lordo | I38 |
| O-OF/Deb Fin | I39 |
| Autof Lordo/AN (=cash flow/attivo) | I40 |
| Deb totali/VA | I41 |
| Deb totali/Ric | I42 |
| PC/ric | I43 |
| Deb finanziari (stimati)/VA | I44 |
| Deb finanziari (stimati)/Ric | I45 |
| Debiti totali/EBITDA | I46 |
| Debiti finanziari/EBITDA | I47 |
| Ebitda-servizio debito/AN | I48 |

Verranno utilizzate le stesse linee di codice, le modifiche riguarderanno i parametri delle funzioni in R e l'obiettivo è ottenere dei modelli previsionali validi anche con questo nuovo dataset.
Di seguito i risultati ottenuti:

## Albero decisionale

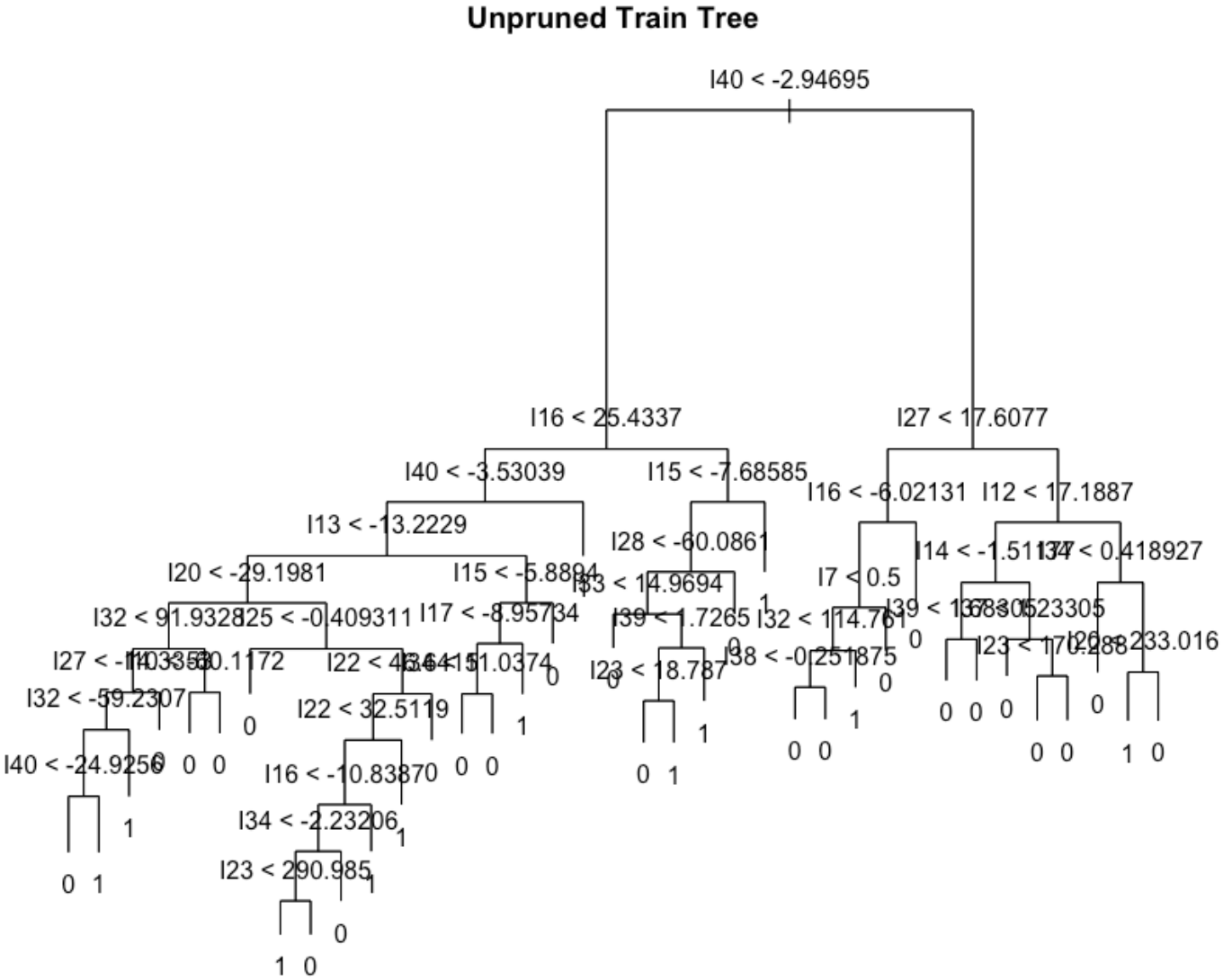
Classification tree:
tree(formula = FLAG ~ ., data = data)
Variables actually used in tree construction:
[1] "I40" "I16" "I14" "I27"

Number of terminal nodes: 8
Residual mean deviance: 0.5285 = 735.2 / 1391
Misclassification error rate: 0.09435 = 132 / 1399




L'output ottenuto è l'albero decisionale generico da cui andremo a costruire il training set e il testing set. Un aspetto interessante è notare come i set di training e di test non necessariamente rispecchino la stuttura e le caratteristiche dell'albero da cui derivano, ne è l'esempio proprio il training tree generato per quest'analisi che mostra una complessità decisamente maggiore rispetto all'albero padre e un errore residuo e un errore di misclassificazione minore.
L'output è il seguente:

Classification tree:
tree(formula = FLAG ~ ., data = data.train)
Variables actually used in tree construction:
[1] "I40" "I16" "I13" "I20" "I32" "I27" "I10" "I25" "I22" "I34" "I23" "I15" "I17"
"I28" "I33" "I39"
[17] "I7"  "I38" "I12" "I14" "I37"
Number of terminal nodes:  37
Residual mean deviance:  0.2359 = 180 / 763
Misclassification error rate: 0.05125 = 41 / 800



L'albero appena generato, considerando il "Misclassification error rate" e il "Residual mean deviance" , mostra un'accuratezza paragonabile ai risultati di algoritmi basati su bootstrap, ma non possiamo considerarlo il risultato finale non essendo un risultato consistente, poiché ottenuto da particolari condizioni del dataset iniziale (in base alle variabili utilizzate) ma anche dalle condizioni del dataset di training generato.
L'obiettivo è generare un modello che prenda quante più informazioni possibili dal training set per poi trovare una relazione che possa dare un'accuratezza soddisfacente nel testing set, è da quello che possiamo effettivamente valutare un risultato.
Applicando la funzione di *accuracy* già vista otteniamo infatti un valore pari a 0.8581636, che possiamo provare a migliorare

## Regressione logistica e Bagging

Per la regressione logistica e il Bagging utilizziamo le stesse funzioni viste prima, con le opportune modifiche relative all'input sul dataset, per il resto non ci sono ottimizzazioni particolari da fare, otteniamo quindi un *accuracy* pari a:

- 0.8915025 per la Regressione Logistica
- 0.8681636 per il Bagging

In questo caso il modello di Regressione Logistica ha un'accuratezza migliore nella previsione rispetto al Bagging.

Analizziamo ora i modelli di Random Forest e Boosting, di cui presteremo attenzione all' ottimizzazione dei parametri.

## Random Forest

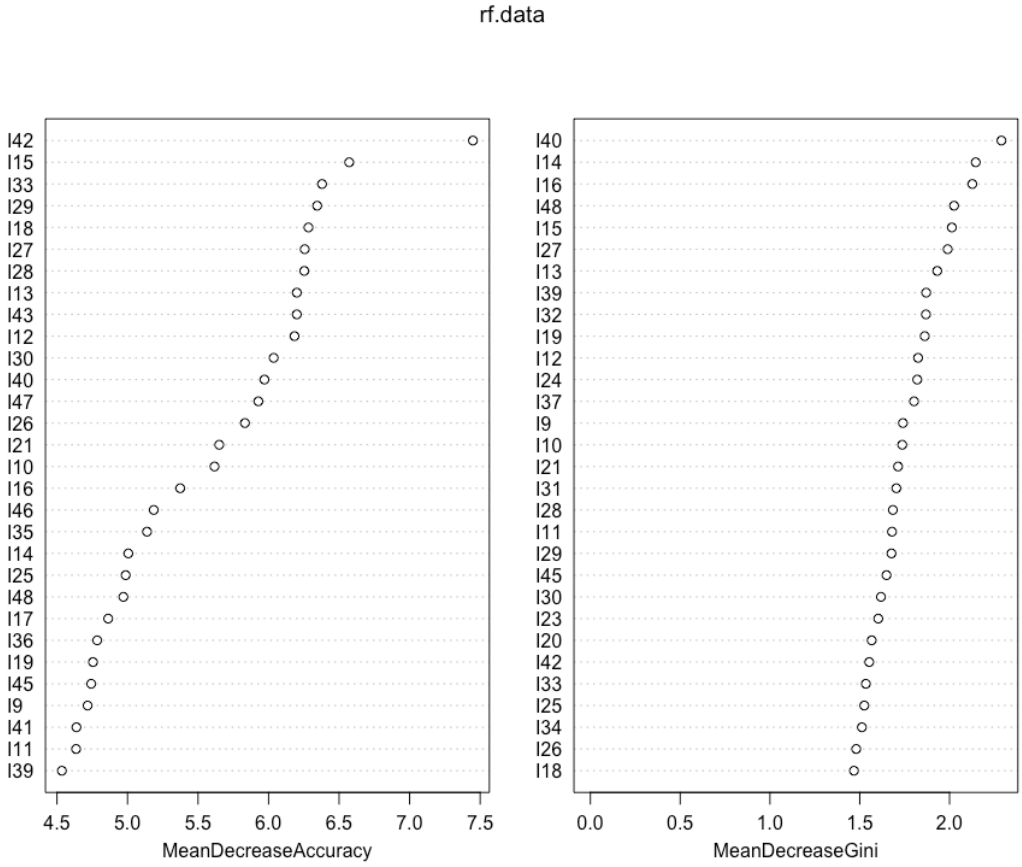
Come spiegato il Random Forest è un caso particolare, ottimizzato, del più generale Bagging. Utilizzando le funzioni viste prima cerchiamo di migliorare il valore di accuracy partendo dalla corretta scelta dei parametri, da cui ricaviamo:

Il valore ottimale di *mtry:*
> data.rf.tune$bestTune
 mtry

1 1

e l'*accuracy* del modello di Random Forest con relativo output grafico dell'importanza delle Variabili,
partendo dalla funzione principale:

*rf.data=randomForest(FLAG~.,data=data.train,*
*mtry=1,importance=TRUE, ntrees=100)*

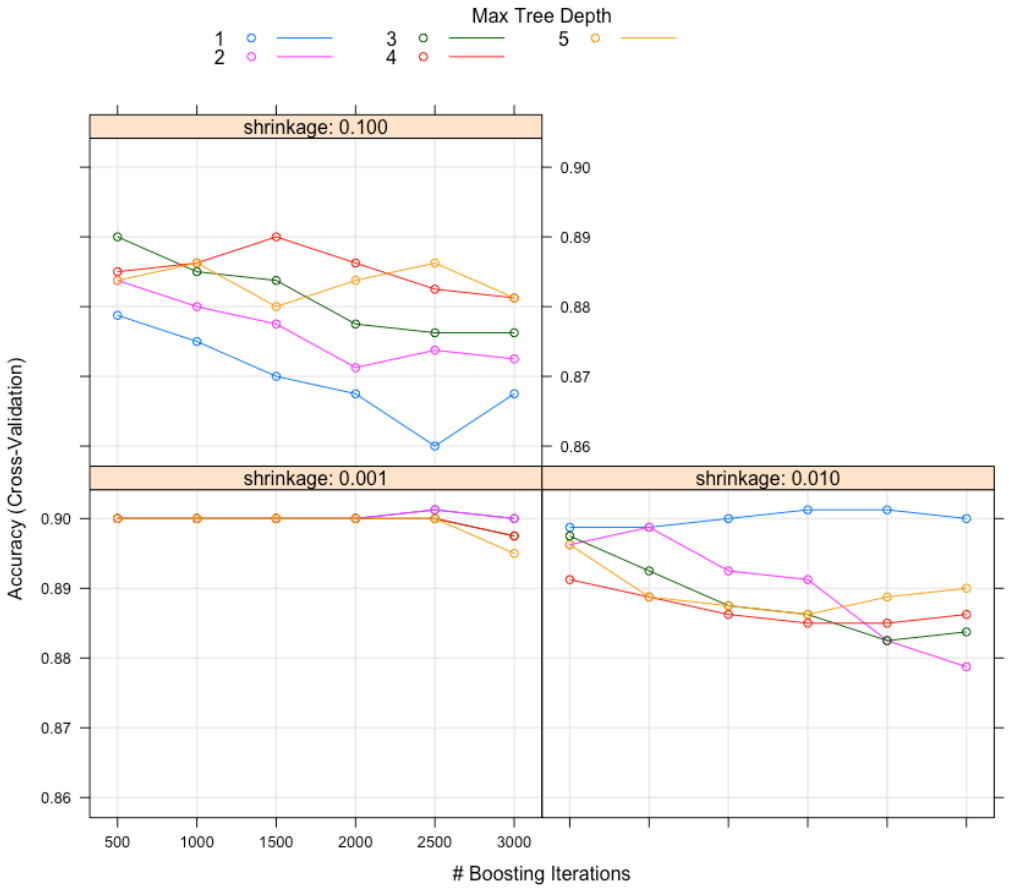



Il valore di accuracy del modello di Random Forest è pari a 0.903172, un buon miglioramento rispetto al Bagging.

## Boosting

Per il Boosting seguiamo lo stesso principio, cerchiamo di ottimizzare i parametri "Max Tree Depth", "shrinkage" e "n.trees" della funzione gbm di Gradient Boosting.

Dalla funzione di “tuning” otteniamo l’output grafico:




e la seguente linea di codice con un recap dei parametri da utilizzare nel modello:

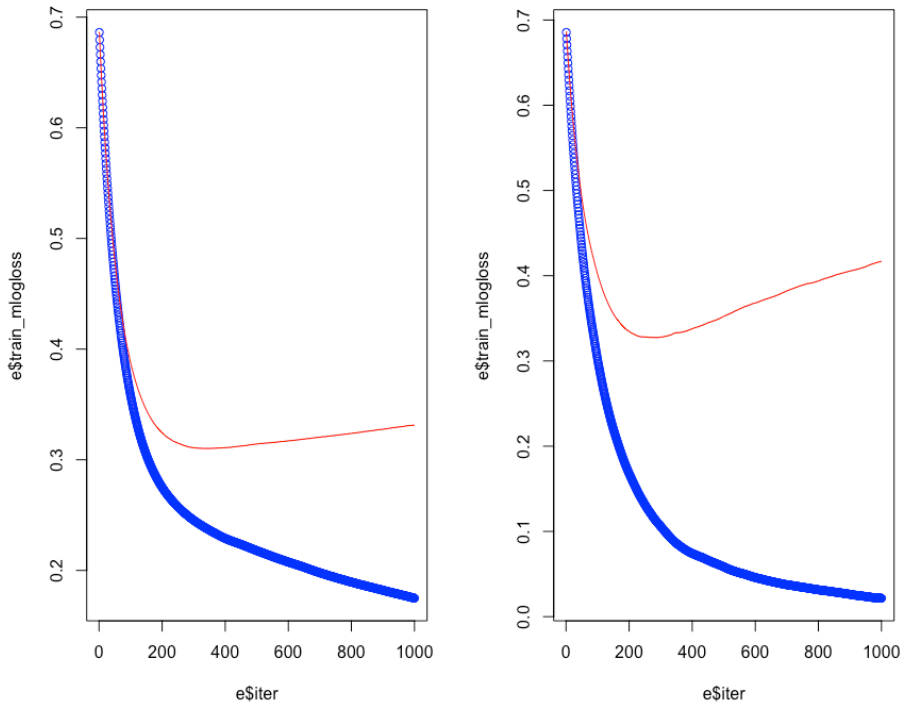
| n.trees | interaction.depth | shrinkage | n.minobsinnode |
|---|---|---|---|
| 2000 | 1 | 0.01 | 10 |

Costruiamo quindi la funzione di boosting con questi parametri ottenendo un *accuracy* pari a 0.8898164.

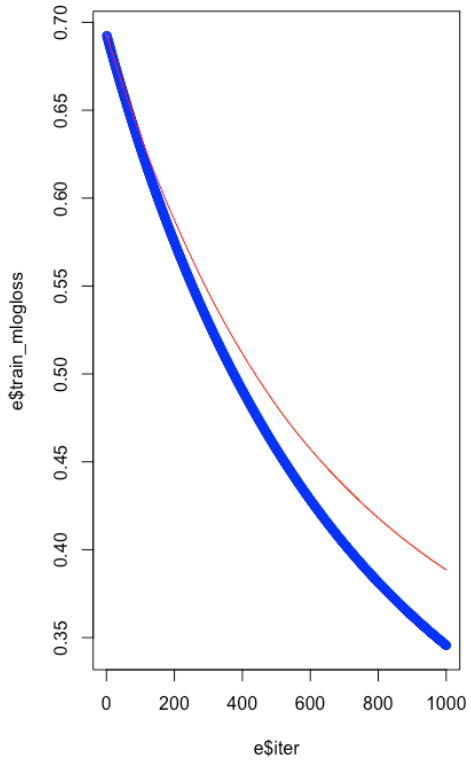
## XGBoost

L’ottimizzazione dell’algoritmo XGBoost si ottiene andando a modificare i valori di “nrounds”, “eta”,”max.depth” e “gamma”.
Cercando i giusti valori dei parametri della funzione *xgb* otteniamo infatti le curve della funzione di costo log loss, rispettivamente per il training data e il testing data, che ci danno un’indicazione della qualità del modello costruito, solo dopo potremo calcolarne l’accuracy.

Analizzando i risultati è importante che le due curve non divergano e che mantengano entrambe un andamento costante decrescente, modificando iterativamente i parametri di controllo, si passa da risultati non soddisfacenti:




al modello definitivo di XGBoost



da cui possiamo poi ricavarne l'accuracy con la solita funzione:

*accuracy=function(actual,predicted) {mean(actual == predicted)}*

ottenendo il valore 0.901182

# Conclusioni

Il presente lavoro di Tesi si è concentrato nello studio e nella realizzazione tramite codice R dei principali algoritmi basati su alberi decisionali, per applicarli poi all'analisi del rischio creditizio aziendale, in particolare per aziende italiane del settore elettronico.
L'obiettivo era ottenere un modello con un'accuratezza elevata e capace di ottenere risultati consistenti, sono stati misurati quindi i valori di accuratezza per più algoritmi e con due dataset inziali diversi, sempre riferiti alle stesse aziende del settore elettronico ma con un'impostazione dati diversa come mostrato nel capitolo 3.
Per entrambe le analisi e per tutti i modelli *decision tree-based* applicati si sono ottenuti dei buoni risultati, qui di seguito un riepilogo:

| | *Analisi 1* | *Analisi 2* |
|---|---|---|
| **Albero Decisionale semplice** | 0.8580702 | 0.8581636 |
| **Regressione Logistica** | 0.8692398 | 0.8915025 |
| **Bagging** | 0.8866871 | 0.8681636 |
| **Random Forest** | 0.8879193 | 0.903172 |
| **Boosting** | 0.8870256 | 0.8898164 |
| **XGBoost** | 0.9082244 | 0.901182 |

Si noti come i risultati siano convincenti anche per le ottimizzazioni che sono state apportate, in entrambe le analisi si evince infatti come i modelli di Random Forest abbiano performato meglio dei modelli di Bagging, allo stesso modo gli algoritmi di XGBoost rispetto a quelli di Boosting.

In particolare nella seconda analisi, dove il dataset era composto da molte meno osservazioni, i risultati ottenuti da Regressione Logistica e Random forest hanno ottenuto risultati paragonabili all'algoritmo XGBoost, il quale rimane comunque tra gli algoritmi più performanti attualmente basati su alberi decisionali.
I valori ottenuti potrebbero essere migliorati, un numero più alto delle osservazioni non necessariamente implica un miglioramento, un'idea però che sicuramente andrebbe a modificare il risultato (e con la dovuta parametrizzazione potrebbe modificarlo in positivo) è l'aggiunta di variabili.
Avere più variabili condiziona tanto l'output degli algoritmi di Bagging e Boosting, quindi di conseguenza anche Random Forest e XGBoost, perché è proprio dall'analisi delle variabili a disposizione che vengono definiti, come spiegato nei paragrafi precedenti, i criteri di split, la "profondità" dell'albero, l'eterogeneità delle classi finali, la velocità di esecuzione dell'algoritmo ecc.
Sicuramente per apportare una modifica ai buoni risultati ottenuti sarebbe consigliato utilizzare variabili con meno correlazione possibile con quelle già utilizzate.
Le applicazioni di un modello ad albero decisionale con un buon livello di accuratezza potrebbe sicuramente migliorare le valutazioni creditizie, rappresentano una validissima alternativa e non a caso le tecniche di Machine Learning stanno prendendo sempre più piede nel settore finanziario.

## Bibliografia e Sitografia


- An Introduction to Statistical Learning with Applications in R, *Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani , 2017*

- Machine Learning with R, *Brent Lantz*, *2013*

- https://cran.r-project.org/

- https://www.rdocumentation.org/

- https://xgboost.readthedocs.io/en/latest/index.html#

- https://statquest.org/