POLITECNICO DI TORINO

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Classificazione di dati sbilanciati e validazione delle performance in un contesto reale

Relatori
Prof.ssa Elena Baralis
Prof. Luca Cagliero
Prof.ssa Silvia Anna Chiusano

Candidato
Sergio Gentile
Matricola: 237010

Anno Accademico 2018/19

Ringraziamenti

Gli impegni lavorativi hanno reso difficile la stesura di questa tesi specialmente nell'ultimo periodo, dove le consegne sono state più stringenti.
Il sostegno di alcune persone è stato fondamentale per poter ultimare gli studi.
In primo luogo ringrazio il professore Luca Cagliero che mi ha fornito un supporto incredibile, dandomi l'opportunità di accrescere la mia conoscenza in un ambito di ricerca da me poco sperimentato ma che ritenevo di fondamentale importanza per la mia crescita accademica e professionale.È doveroso ringraziarlo anche per la grandissima disponibilità concessami, sia in termini accademici ma anche per essersi mostrato sempre disponibile alle mie esigenze lavorative.
Un ringraziamento va anche alla professoressa Elena Baralis ed alla professoressa Silvia Anna Chiusano che mi hanno dato l'opportunità di intraprendere questo progetto accontentando le mie richieste.
Voglio ringraziare la mia famiglia, che cinque anni fa mi ha dato la possibilità di vivere qua a Torino, investendo su di me e sul mio futuro.
Un ringraziamento anche ai miei nonni che oggi non sono presenti ma che mi hanno fornito tantissimo supporto.
Anche se al tempo sembrava impossibile, oggi sono felice di avercela fatta ed avervi resi orgogliosi di me.
Ringrazio specialmente mia madre Elisa, per l'enorme pazienza nel rileggere innumerevoli volte questa tesi e aver corretto il mio pessimo italiano.
Un grazie speciale va a tutti i ragazzi del Collegio Einaudi, dal primo all'ultimo.
Anche se tra studio e lavoro sono stato poco presente, i più vecchiotti sanno che in questi anni mi sono trovato benissimo con voi ed in certe occasioni siete stati la mia forza all'incubo chiamato "sessione" ed ai periodi universitari più difficili.
Inoltre vi ringrazio tutti per una cosa fondamentale: è proprio qui che ho capito che per far bene nella vita bisogna studiare tanto e sacrificarsi per realizzare quello che si vuole, apprendendo tutto questo da gente molto più brava di me a cui mi sono ispirato.
Può sembrare una cosa banale, ma probabilmente se non mi avessero accettato in collegio starei ancora sudando per la laurea triennale, perchè in fondo "che triennale è se non dura sette anni?"
Tra tutti, un ringraziamento particolare va ad Angela per avermi permesso di ultimare questa tesi e non solo: grazie per avermi aspettato nell'ultimo mese fino a notte fonda, aiutandomi a completare un lavoro che tra impegni lavorativi ed altro sembrava impossibile da finire, probabilmente non mi sdebiterò mai.
Grazie a tutti i miei amici della Sicilia, l'inizio di quest'avventura (anche se è passato tanto tempo) non sarebbe stato lo stesso e grazie ad Ismaele per essere ancora mio amico dopo diverso tempo ed esserci sempre.
Un grazie infinite a tutti.

Il Vostro gentile Sergio
Torino, 16 Aprile 2019

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

L'analisi dei dati è da sempre importante in campo scientifico e non, in quanto permette l'estrazione di informazioni intrinseche presenti all'interno di una collezione di dati sottoposta ad analisi.
Ad esempio, il reparto di marketing di una grande azienda che ha a disposizione un'elevata quantità di informazioni relative alle vendite degli ultimi anni in un determinato settore di mercato, può decidere attraverso essa se sarà conveniente intraprendere o meno una certa strategia di mercato.
La tecnologia fornisce un forte aiuto nell'automatizzare il processo d'analisi.
Il *Machine Learning* offre una serie di algoritmi in grado di supportare l'apprendimento automatico, consentendo alle macchine di acquisire conoscenza tramite la costruzione di opportuni modelli.
Nell'ambito del Machine Learning, la *classificazione* rappresenta uno dei problemi più ricorrenti. Quest'ultima consiste nell'identificazione della classe di appartenenza di un'entry sulla base della conoscenza appresa tramite la collezione di dati sulla quale è stato costruito il modello predittivo.
Un esempio rilevante di classificazione può esser trovato nel contesto dell'analisi di dati veicolari, dove si vuol effettuare un'identificazione preventiva dei guidatori ad alto rischio di sinistro stradale. La predizione si baserà sulle caratteristiche del veicolo assicurato, sui dati anagrafici del titolare della polizza assicurativa, sulle informazioni relative al suo stile di guida e sulle percorrenze effettuate in passato.
Negli ultimi anni, data la crescita esponenziale della quantità e della velocità con cui vengono raccolti e immagazzinati ampi flussi di informazioni, le tecniche di Machine Learning sono applicate su collezioni di dati di ampio volume, chiamate *Big Data*.
In tale contesto, la loro gestione richiede il supporto di un ambiente distribuito, in quanto viene meno la possibilità di memorizzare e processare i Big Data in un ambiente centralizzato. Questo comporta, a seguito dello sviluppo di tecniche apposite per la classificazione, un'analisi di complessità atta ad analizzare la quantità di risorse necessarie per riportare gli sviluppi della propria ricerca in un ambiente reale.
In letteratura si possono identificare, sulla base del modello matematico utilizzato, diversi algoritmi di classificazione: *Decision Tree, Random Forest, Support Vector Machines, Naive Bayes Classifier, Reti Neurali* e *Classificatori Associativi*.
Nei casi di studio reali, non ci si limita alla semplice scelta di uno degli algoritmi sopra elencati, ma si procede con un'analisi più profonda atta ad individuare la strategia più idonea in grado di gestire il dataset sotto esame.
Una problematica comune da affrontare nel contesto della classificazione è legata alla gestione di *dataset sbilanciati* (*Imbalanced Dataset*) rispetto ad una classe.
Per Imbalanced Dataset si intende una tipologia di collezione di dati in cui il numero di entry di una classe prevale di gran lunga sull'altra (maggiore è la differenza tra il numero di entry della classe maggioritaria e minoritaria, più alta sarà la complessità nella classificazione).

Lo sbilanciamento tra classe maggioritaria e minoritaria viene stabilito attraverso la msiura *Imbalanced Ratio (IR)*, definita come il rapporto tra numero di entry della classe maggioritaria e minoritaria.
In questo contesto, la predizione risulta problematica poiché un modello di classificazione tende ad adattarsi alla classe prevalente (fenomeno dell'overfitting), il che richiede una gestione mirata alla risoluzione del problema.
Esempi di dataset sbilanciati si trovano nella vita di tutti i giorni: si immagini di voler determinare tramite il Machine Learning se un paziente è portatore o meno di una certa malattia rara. Questo implica che il dataset a disposizione per la costruzione del modello conterrà un alto numero di entry corrispondenti a dati fisiologici di pazienti sani ed un bassissimo numero di dati inerenti ad un portatore della malattia, il che favorirà la costruzione di un classificatore indotto a predire, a prescindere dall'entry in ingresso che necessita di classificazione, sempre la classe di maggioranza.
Tale problematica, oltre ad avere valenza dal punto di vista scientifico, diventa molto importante in un contesto reale, nel quale classificare un paziente malato come sano indurrebbe a non fornirgli le giuste cure.
Data l'importanza della tematica, in letteratura sono presenti diverse metodologie che mirano a superare la problematica dello sbilanciamento tra classi tramite analisi statistiche ed algoritmi ad hoc.
Si possono identificare due prime macro-categorie di soluzione: l'Oversampling e l'Undersampling.
L'*Oversampling* consiste in una replica casuale dei dati della classe minoritaria.
Una particolare tipologia di Oversampling è lo *SMOTE*, che consiste nella creazione di nuovi dati appartenenti alla classe minoritaria a partire da quelli già esistenti.
L'*Undersampling* si basa, invece, sull'eliminazione casuale delle entry appartenenti alla classe maggioritaria.
Le due tipologie di approccio seguono strade diametralmente opposte: tuttavia entrambe presentano la peculiarità di lavorare solamente a livello di dato (*Data Level Approach*), senza intaccare in alcun modo la porzione di codice relativa alla classificazione.
È possibile individuare due ulteriori macro-categorie che in questo caso agiscono a livello di codice (*Algorithm Level Approach*): il *Bagging* e l'*Ensemble*.
Con la tecnica del *Bagging* vengono creati n modelli del classificatore originale, ognuno dei quali sarà addestrato con una versione differente del training set originale.
L'*Ensemble*, allo stesso modo del Bagging, creerà diversi modelli di classificazione, ma in questo caso a cambiare non sarà il training set ma il classificatore stesso.
Entrambe le metodologie utilizzano un meccanismo a voto di maggioranza per determinare la classe d'appartenenza dell'entry da classificare.
Gli approcci dell'Oversampling e dell'Undersampling risultano funzionali in quanto hanno il grande vantaggio di ridurre l'IR del dataset sbilanciato, oltrepassando il problema dell'overfitting.
Inoltre, il Bagging e l'Ensemble supportano un miglioramento della decisione intrapresa, attraverso l'utilizzo di diversi classificatori.

Entrambe le macro-categorie hanno però lo svantaggio di lavorare in maniera casuale: nel caso dell'Oversampling e dell'Undersampling i dati vengono duplicati o eliminati in maniera casuale. Lo SMOTE cerca di superare tale problematica, ma non si ha nessuna garanzia che il dato creato apparterrà alla classe minoritaria; nel caso del Bagging e dell'Ensemble la scelta dei parametri che compongono i diversi classificatori e la formazione dei diversi training set avviene in modo casuale.
L'obiettivo di questo lavoro di tesi è lo studio del problema della classificazione di dati sbilanciati attraverso tecniche basate su clustering, campionamento dei dati (Oversampling e Undersampling) e modelli associativi. Un studio preliminare della letteratura consentirà di identificare le tecniche allo stato dell'arte e i loro relativi vantaggi e svantaggi. Successivamente, verranno proposte delle estensioni delle tecniche esistenti finalizzate a migliorare la qualità della predizione dei campioni della classe minoritaria.
In una prima fase di studio, il Clustering fornirà un aiuto fondamentale nella comprensione degli elementi da duplicare o eliminare, rispettivamente nei casi dell'Oversampling e dell'Undersampling.
In un secondo momento, verrà proposta una tecnica che consente la valutazione degli elementi sintetizzati dallo SMOTE, al fine di scartare i dati creati come appartenenti alla classe minoritaria ma che in realtà fanno parte dell'altra classe.
La totalità delle soluzioni presentate verranno validate su un caso di studio reale e su diversi dataset di Benchmark.
In conclusione, verrà dimostrato che le nuove metodologie sviluppate consentiranno di ottenere dei risultati migliori rispetto agli approcci già presenti in letteratura, in termini di *Precision, Recall* ed *F1-Score* della classe minoritaria.

2 Il processo di Data Mining e Knowledge Discovery

Con *Data Mining* si intende un processo composto da una serie di step per la quale vengono utilizzate diverse tecniche atte ad estrarre conoscenza da una grande mole di dati.

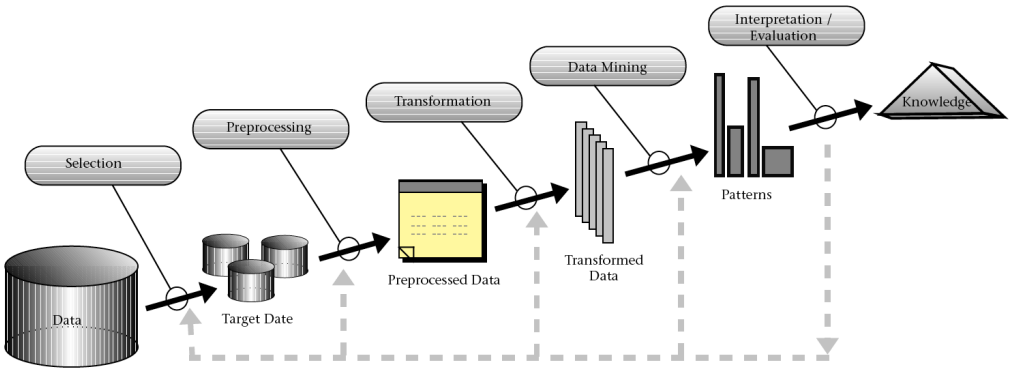


Figura 2.1: Processo d'analisi dei dati [1]

La Figura 2.1 riassume i principali passi che compongono il Data Mining:

1. ***Selection:*** processo in cui vengono selezionate solamente le informazioni d'interesse.
   Solitamente questo primo step viene effettuato insieme ad un esperto di dominio, capace di identificare un sottoinsieme di features rilevanti ai fini dell'analisi;

2. ***Preprocessing:*** prima di applicare un qualsiasi algoritmo di classificazione è opportuno elaborare i dati di partenza (*dati grezzi*) al fine di gestirne la mancanza, effettuarne la normalizzazione, la discretizzazione ed altro;

3. ***Transformation:*** si intende il processo per il quale il dataset viene trasformato da un formato ad un altro.
   Nella maggior parte dei casi, è opportuno modificare il dataset su cui si è effettuato il preprocessing a seconda dell'analisi che vogliamo effettuare su di esso.
   Un esempio di trasformazione può essere l'aggregazione dei dati da una granularità temporale (e.g. giorni) ad un'altra (e.g. anno);

4. ***Applicazione algoritmi di Machine Learning:*** partendo dal dataset trasformato, sarà possibile utilizzare degli algoritmi di Machine Learning capaci di assegnare a dati non classificati delle etichette di classe;

5. ***Interpretation/Evaluation:*** i dati predetti verranno opportunamente confrontati con i risultati attesi al fine di comprendere la qualità del processo di classificazione.

Per alcuni algoritmi che generano modelli interpretabili (e.g. alberi di decisione, regole d'associazione) sarà possibile estrarre delle informazioni utili intrinseche nel dataset sottoposto all'analisi che normalmente un esperto di dominio non è capace di cogliere.

2.1 L'apprendimento

La costruzione di un modello di classificazione parte da una fase chiamata *training* che consente, a partire dai dati in nostro possesso, la definizione di un modello matematico capace di catalogare delle entry date in input al classificatore.
In letteratura vengono definite due tipologie di apprendimento:

- ***Supervisionato:*** il modello viene costruito partendo da un training set dove ad ogni input è associata la corrispondente etichetta di classe.
  In tal modo, definendo un insieme di input e di output, tramite un'opportuna funzione d'errore, viene confrontata la risposta in uscita con quella attesa, apportando così le opportune correzioni al modello.
  Un tipico esempio di apprendimento supervisionato sono le *Reti Neurali (NN)*, dove viene applicato un algoritmo di *backpropagation* al fine di modificare i pesi che compongono le connessioni del modello.

- ***Non Supervisionato:*** rispetto al modello supervisionato, le entry del training set non contengono le etichette di classe.
  Pertanto, l'apprendimento viene effettuato cercando di creare un modello capace di estrarre conoscenza "nascosta" all'interno del dataset.
  Tipici esempi di apprendimento non supervisionato sono il Clustering e le Regole d'associazione.

Nel corso di questa tesi verrà utilizzata la classificazione supervisionata ma essa sfrutterà metodologie non supervisionate per un miglioramento delle performance.

2.2 La classificazione

Con classificazione definiamo il processo per cui ad ogni entry non etichettata del dataset viene assegnata un'etichetta di classe.
Immaginiamo di essere in possesso di un set di dati formato da diverse righe R e colonne C, per il quale si vuole conoscere l'output Y per ogni riga R_i.
La classificazione può esser vista come una sequenza di operazioni capaci di:

1. Prendere in input la riga R_i;
2. Elaborare R_i attraverso un modello di classificazione costruito grazie ad una precedente fase di training (vedi Sezione 2.1);
3. Ritornare in output un'etichetta di classe Y.

Alla fase di classificazione, precede una fase di *training*, utile per costruire il modello. Sarà proprio la fase di training esposta nella Sezione 2.1 che predisporrà il modello ad una corretta classificazione.

2.2.1 Validazione del modello

A seguito dell'applicazione di un algoritmo di classificazione su di un dataset, sarà importante valutare delle misure di qualità al fine di comprendere le performance del modello sviluppato.
Chiaramente questo sarà possibile se la porzione di dati per cui è stata determinata l'etichetta di classe incognita in realtà ne è provvista, rendendo possibile un confronto tra *classe predetta - classe reale*.
Si rimanda alla Sezione 2.2.2 nella quale viene esposto in modo più tecnico la metodologia con la quale si effettua tale confronto.
In questa prima parte si vuol far capire quali sono le misure di qualità d'interesse, la cui base di partenza è la *matrice di confusione* (Tabella 2.1).

Tabella 2.1: Matrice di confusione

Reale/Predetta	**Positivo**	**Negativo**
Positivo	TP	FP
Negativo	FN	TN

Supponiamo di voler classificare, a seguito della fase di training, delle entry con etichette di classe *"Positivo"* e *"Negativo"*.
Definiamo:

- ***True Positive (TP):*** numero di entry appartenenti alla classe *"Positivo"* correttamente predette;
- ***True Negative (TN):*** numero di entry appartenenti alla classe *"Negativo"* correttamente predette;
- ***False Positive (FP):*** numero di entry appartenenti alla classe *"Negativo"* ma a cui è stata assegnata l'etichetta di classe *"Positivo"*;
- ***False Negative (FN):*** numero di entry appartenenti alla classe *"Positivo"* ma a cui è stata assegnata l'etichetta di classe *"Negativo"*.

La matrice di confusione dà un'idea dell'andamento della classificazione, partendo dai valori letti dalle sue celle.
Combinando opportunamente tali valori, sarà possibile estrarre le misure di qualità con le quali valutare la bontà del modello:

- ***Accuratezza:*** indica la percentuale di dati classificati correttamente:

$$A = \frac{TP + TN}{TP + TN + FP + FN}$$

Tale misura tuttavia potrebbe non dare una buona indicazione di quanto l'algoritmo sia effettivamente performante.
Questa considerazione è vera nel caso in cui le performance predittive legate ad una classe (Classe A) sono nettamente migliori rispetto a quelle di un'altra classe (Classe B), dove inoltre il numero di entry della Classe A è molto maggiore del numero di entry della Classe B.
Poiché la misura è indipendente dalla quantità di elementi all'interno di una classe, non si potrà capire quanto effettivamente l'algoritmo sarà capace di predire bene la classe minoritaria.

- ***Precisione:*** indica la percentuale di dati per cui è stata predetta la classe in modo corretto rispetto al numero totale di dati predetti con tale classe:

$$P = \frac{TP}{TP + FP}$$

- ***Richiamo:*** indica la percentuale di dati per cui è stata predetta la classe in modo corretto rispetto al numero totale di dati appartenenti a tale classe:

$$R = \frac{TP}{TP + FN}$$

- ***F1-Score:*** rappresenta la media armonica tra precisione e richiamo:

$$F1\text{-}Score = 2 \cdot \frac{P \cdot R}{P + R}$$

Poiché si vuole massimizzare sia il richiamo che la precisione, questa misura è ottima per capire quali sono le vere performance del nostro classificatore.

2.2.2 Training Set e Test Set

Ai fini delle valutazioni sulle performance predittive, è utile disporre di due differenti porzioni di dati:

- ***Training Set:*** porzione del dataset che viene utilizzata per addestrare il classificatore;
- ***Test Set:*** sottoinsieme più piccolo utilizzato per la validazione del modello.

L'idea è validare il modello sulla base di dati per la quale in realtà la classe è già ben definita.
Al fine di disporre dei due sottoinsiemi sopra elencati, possono essere utilizzate diverse tecniche di splitting.

Prima di esporle, bisognerà definire il concetto di *Fold*.
Data una collezione di dati in ingresso, è possibile suddividerla in *K-Fold* differenti in modo tale da creare K diversi sottoinsiemi che verranno poi utilizzati nella fase di training e validazione.
Definito il concetto di Fold, è ora possibile passare alla definizione delle tre tecniche di validazione utilizzate in questa tesi:

- ***Holdout:*** il training set e il test set vengono generati dividendo secondo una certa percentuale il dataset di input.
  A titolo d'esempio, il 70% delle entry a nostra disposizione può essere utilizzato per comporre il training set, mentre il restante 30% comporrà il test set;

- ***Cross Validation:*** a seguito della generazione di *K-Fold* differenti, tra i k sottoinsiemi il *k-esimo* viene considerato come test set, mentre tutti gli altri sottoinsiemi restanti andranno a comporre il training set.
  La grande differenza rispetto all'Holdout sarà l'iterare k volte il processo di testing, facendo variare di volta in volta il sottoinsieme che verrà utilizzato come test set.
  In tal modo tutti gli elementi del dataset apparterranno sia al training set che al test set.
  Quindi, il grande pregio offerto dalla Cross Validation consiste nella possibilità di validare la propria soluzione sull'intero dataset piuttosto che su di un piccolo subset come fatto dall'Holdout, facendo così entrare nel dataset di test tutti i dati della collezione;

- ***Stratified Cross Validation:*** viene utilizzata la normale Cross Validation, ma i *K-Fold* vengono generati in modo tale da avere in ogni Fold la medesima percentuale di elementi della stessa classe.
  Ciò è molto utile quando si lavora con dataset sbilanciati, dove si vuol garantire sia in fase di training che di test una certa percentuale di elementi della classe minoritaria;

- ***Leave One Out Validation:*** è una variante della Cross Validation nella quale il *k-esimo* Fold di test è composto da un solo elemento della collezione di dati.
  Questo implica che ad ogni passo della Leave One Out Validation verranno utilizzati *n-1* elementi per comporre il training set, mentre un solo elemento farà parte del test set.
  Tale processo di validazione è particolarmente utile quando si lavora con dataset abbastanza piccoli ($\simeq$ 1000 elementi).

3 Modelli supervisionati e non supervisionati

Nella Sezione 2.1 è già stata introdotta la distinzione tra modello supervisionato e non. In questo capitolo si vuol entrare maggiormente nel dettaglio di quali sono gli algoritmi che compongono i due macro-gruppi, spiegandone l'implementazione e l'utilizzo.
Si precisa che il concetto di non supervisionato non è legato unicamente a quello di classificazione. Ad esempio, nel caso del Clustering non viene assegnata alcuna etichetta di classe ai dati che compongono il dataset, ma il processo si limita ad includere ogni dato ad un gruppo, a seconda delle caratteristiche di cui è composto.
È necessario parlare di non supervisionato poiché in fase di applicazione dell'algoritmo i gruppi d'appartenenza non sono noti, ma verranno definiti solamente a seguito del processo di clustering.

3.1 La metodologia non supervisionata

Seppur non utilizzati a scopo predittivo, saranno sfruttati algoritmi non supervisionati come supporto per la classificazione supervisionata.
In tal senso, verranno utilizzate due metodologie:

- *Clustering*: consente di individuare per un dato un gruppo d'appartenenza a seconda delle caratteristiche che lo compongono.
  Questo sarà utile per effettuare un Oversampling o Undersampling mirato alle caratteristiche del dato in esame;

- *Regole d'associazione*: vengono estratte dal dataset delle informazioni sotto forma di regole.
  Ciò sarà utile per l'implementazione del classificatore associativo.

3.1.1 Il Clustering

All'interno della metodologia non supervisionata, gli algoritmi di clustering si pongono come obbiettivo il raggruppamento, dentro il medesimo cluster, di dati simili tra loro (il concetto di *simile* verrà definito in seguito).
In letteratura, tali algoritmi vengono divisi in due categorie: *partizionale* e *gerarchico*.
Nel *clustering partizionale* ogni dato del dataset potrà appartenere ad uno ed un solo cluster.
Al contrario, nel *clustering gerarchico*, sarà possibile associare un punto a più cluster.
In questo ultimo caso possiamo quindi immaginare i cluster organizzati sotto forma di albero.
Poiché un cluster è composto da una serie di dati, è importante trovare un dato che ben rappresenta l'intero gruppo.
Il rappresentante verrà scelto tra il *centroide* ed il *metoide*:

- ***Centroide:*** tra i dati della collezione viene selezionato il dato più vicino al centro del cluster.
  Il centroide sarà quindi un punto appartenente alla collezione;

- ***Metoide:*** rappresenta la media tra tutti i dati di un cluster.
  Al contrario del centroide, non è garantito che il metoide sia un dato appartenente alla collezione di dati, ma potrebbe essere un dato fittizio.

La definizione di centroide e metoide nasce dalla necessità di dover confrontare due o più cluster tra loro, ad esempio per misurarne la similarità.
Possono essere definite diverse misure, utilizzate dagli algoritmi che costruiscono i cluster:

- ***Min:*** viene selezionata la distanza minima tra due punti di cluster diversi;
- ***Max:*** viene selezionata la distanza massima tra due punti di cluster diversi;
- ***Distanza media:*** dati i cluster A e B, viene calcolata la media tra tutte le distanze dei punti $p_i \in A$ e $q_i \in B$;
- ***Distanza tra centroidi o metoidi:*** viene calcolata la distanza euclidea tra i centroidi o i metoidi dei cluster A e B.

Nel corso di questa tesi verranno utilizzati due algoritmi di clustering: il **K-Means** e il **DBSCAN**.
Nelle Sezioni 3.1.2 e 3.1.3 è possibile trovare una descrizione ad alto livello dei due algoritmi.

3.1.2 K-Means

Il **K-Means** è un algoritmo di clustering di tipo partizionale.
Esso ha la caratteristica di utilizzare come punto rappresentativo del cluster il centroide ed assegnare ogni punto del dataset al centroide più vicino.
L'utilizzo del K-Means presuppone una scelta a priori del parametro K, rappresentante il numero di cluster che verranno generati a seguito della sua esecuzione.
Lo pseudocodice dell'algoritmo è il seguente:

1. Selezione del valore K;
2. Posizionamento dei K centroidi nello spazio in maniera casuale;
3. Assegnazione di ogni punto della collezione di dati al centroide più vicino.
   La distanza *centroide-punto* tipicamente viene calcolata attraverso la definizione di distanza euclidea;
4. Valutazione del centroide di ogni cluster;
5. Ripetizione dell'algoritmo a partire dal passo 3, fintanto che i valori dei centroidi non rimangano invariati o comunque si discostino per un errore (piccolo) definito dall'utente.

Un esempio di iterazione può essere data dalla rappresentazione nella Figura 3.1:

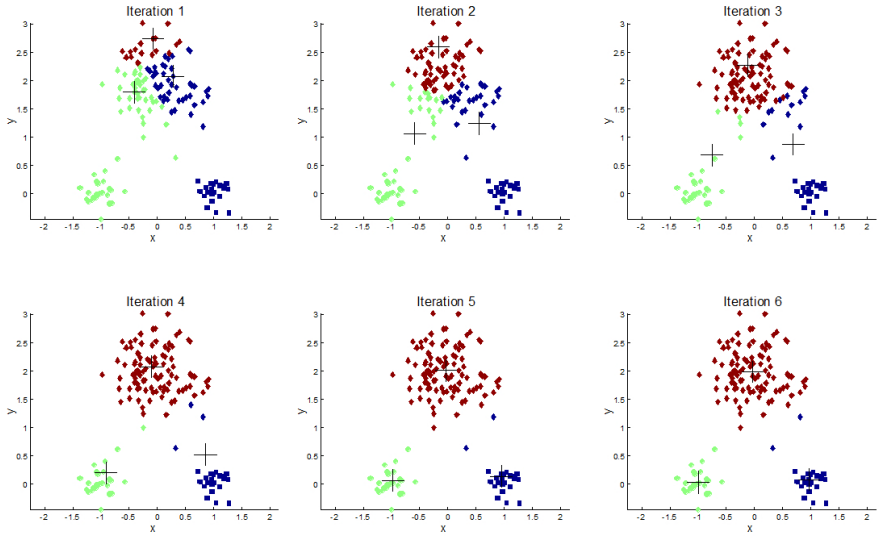


Figura 3.1: Iterazioni del K-Means [2]

Durante la prima iterazione (Figura 3.1, Iteration 1), i centroidi vengono posizionati in maniera del tutto casuale.
Proseguendo, nel corso delle iterazioni, notiamo che i centroidi si spostano fino ad assumere la posizione desiderata.

Il forte limite del K-Means risiede nella scelta del parametro K:

- Quale sarà il valore del parametro K?
  Nel nostro esempio, scegliere un valore di K diverso da tre, significa non individuare nel modo corretto i cluster (probabilmente un quarto cluster si andrebbe a posizionare all'interno del cluster di colore rosso);
- Potenzialmente diverse esecuzioni dell'algoritmo possono portare a risultati diversi, data la casualità con cui vengono posizionati i K centroidi iniziali;
- Lavora bene con cluster di forma circolare.
  Se i cluster assumono forme differenti, l'algoritmo tenderà ad agglomerare all'interno di un cluster tutti i suoi punti vicini, non badando alla forma e alla densità.

3.1.3 DBSCAN

Per superare i limiti di una clusterizzazione dalla forma circolare e di una conoscenza a priori del numero di cluster che vorranno essere identificati, potrà essere utilizzato il DBSCAN.
Esso è un algoritmo *density-based*, ovvero che basa il proprio funzionamento sul concetto di vicinanza tra punti della collezione di dati.
Per permetterne il corretto funzionamento, dovranno essere definiti i seguenti parametri:

- ***Eps:*** indica il raggio entro il quale verranno considerati i punti che parteciperanno attivamente al Clustering;
- ***MinPts:*** definisce una soglia secondo la quale verranno classificati i punti che concorreranno alla creazione del cluster.

In base al numero di punti all'interno del raggio *Eps*, un punto della collezione di dati potrà essere identificato come:

- ***Core Point:*** all'interno di *Eps* si trovano una quantità di punti maggiore o uguale a *MinPts*:
- ***Border Point:*** all'interno di *Eps* non viene trovata una quantità di dati necessari per cui il punto potrà essere etichettato come *core point*, ma si trova in ogni caso nel vicinato di un *core point*;
- ***Noise Point:*** non viene soddisfatta nessuna delle due condizioni precedenti.

Lo pseudocodice dell'algoritmo è il seguente:

1. Vengono eliminati i *noise point*;
2. Viene assegnata la variabile $current\_cluster\_label = 1$;
3. Per ogni *core point i-esimo* non ancora etichettato, in primo luogo viene incrementata la variabile $current\_cluster\_label = current\_cluster\_label+1$, successivamente si assegna $current\_cluster\_label$ al *core point* preso in considerazione;
4. Ogni punto all'interno di *Eps*, se sprovvisto dell'etichetta di cluster, viene etichettato con il valore corrente di $current\_cluster\_label$;
5. Il procedimento tornerà al punto 3 finché tutti i punti non saranno etichettati.

Il DBSCAN lavora bene nel caso di collezioni di dati come quella presentata nella Figura 3.2:

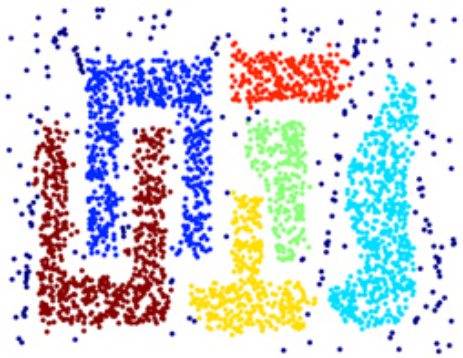


Figura 3.2: Clustering con DBSCAN [2]

Si può notare che vengono gestiti perfettamente cluster con forme differenti.
In tale situazione il K-Means avrebbe inserito i punti all'interno di cluster circolari, non badando a quale fosse la reale forma del cluster.
Questo accade perché il K-Means basa il proprio funzionamento sul solo concetto di similarità, tralasciando il concetto di densità.
Quali sono le problematiche riscontrate con l'utilizzo del DBSCAN?

- È difficile capire il valore da assegnare ad *Eps* e *MinPts* per far funzionare al meglio il DBSCAN. Valori anche poco diversi di *Eps* e *MinPts* possono portare a risultati differenti;
- Soffre con dataset a densità variabile.

3.1.4 Regole d'associazione

A titolo d'esempio, consideriamo il seguente dataset (Tabella 3.1) [3]:

Tabella 3.1: Dataset delle transazioni

TId	**Items**
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diapers, Milk
4	Beer, Bread, Diapers, Milk
5	Coke, Diapers, Milk

Esso rappresenta una serie di transazioni ed i rispettivi prodotti acquistati, in particolare:

- **TId**: indica l'ID della transazione;
- **Items**: contiene la lista dei prodotti associati ad una transazione.

Definiamo una *regola d'associazione* [4] come un'implicazione nella forma

$$X \implies Y$$

dove:

- X viene definito come *corpo* della regola.
  Questa parte corrisponde al cosiddetto *itemset*, ovvero un set di *item*;
  Seguendo il dataset della Tabella 3.1, un esempio di itemset potrebbe essere $X = \{Beer, Diapers\}$.
- Y viene definita come *testa* della regola.
  Essa ci dà un'indicazione del tipo: insieme all'itemset $X = \{Beer, Diapers\}$ è stata trovata la testa $Y = \{Milk\}$.

È importante notare che il simbolo $\implies$ indica solamente la *co-occorrenza* tra il corpo e la testa della regola, non indica in alcun modo *causalità*.

3.1.4.1 Misure di qualità per le regole d'associazione

Immaginiamo di aver estratto, sfruttando un opportuno algoritmo, la seguente regola:

$$Milk \implies Diapers$$

Il significato intrinseco di questa espressione indica che esistono una serie di transazioni all'interno del dataset della Tabella 3.1 per cui chi ha acquistato del latte ha acquistato anche dei pannolini.
In generale tutti gli algoritmi d'estrazione di regole estrapolano solamente le regole che soddisfano certi criteri di qualità che vengono definiti dalle seguenti misure:

- ***Supporto:*** indica la percentuale di transazioni che contengono sia X che Y:

  $$sup(A, B) = \frac{\#\{A, B\}}{|T|}$$

  Capiamo che il supporto ci dà un'indicazione di quante volte troviamo all'interno del dataset l'associazione X, Y, ma non risponde alla domanda: data X, qual è la probabilità di trovare Y?
  La seguente misura risponde a questa domanda.

- ***Confidenza:*** indica, date le transazioni che contengono X, qual è la percentuale di esse che contengono Y:

  $$conf(A, B) = \frac{sup(A, B)}{sup(A)}$$

  Essa quindi indica la probabilità condizionata di trovare Y avendo trovato X, dando un'indicazione di quanto sia forte il legame $X \implies Y$.

- ***Lift o Correlazione:*** indica il livello e la tipologia di correlazione espressa dalla regola:

  $$lift(A, B) = \frac{P(X, Y)}{P(X)P(Y)} = \frac{conf(X, Y)}{sup(Y)}$$

  Per le regole correlate positivamente, il lift assumerà un valore maggiore di 1, per le regole negativamente correlate un valore minore di uno.
  Se il lift assume valore pari a 0, esso indica indipendenza statistica tra la testa e il corpo della regola.

A titolo esaustivo, si riportano degli esempi di calcolo utilizzando il dataset delle transazioni della Tabella 3.1. Data la regola

$$R : Milk \implies Diapers$$

Si hanno delle misure di qualità pari a:

$$sup(R) = \frac{\#\{Milk, Diapers\}}{\#\{Milk\}} = \frac{3}{5} = 60\%$$

$$conf(R) = \frac{\#\{Milk, Diapers\}}{|T|} = \frac{3}{4} = 75\%$$

$$lift(R) = \frac{conf(R)}{sup(Diapers)} = \frac{0.75}{0.60} = 1.25$$

Le misure di qualità sono utili non solo per valutare la qualità delle regole estratte, ma anche per l'estrazione stessa.
Algoritmi d'estrazione di regole come *APRIORI* [5] e *FP-Grouth* [6] utilizzano soglie di supporto e confidenza minima così da lavorare e presentare in output solamente un set ridotto di regole.
Se non si utilizzassero tali soglie, la quantità di regole estratte potrebbe essere spropositata, soprattutto per collezioni di dati di grandi dimensioni.

3.2 La classificazione supervisionata

Nel corso di questa tesi verranno utilizzati diversi algoritmi di classificazione supervisionata [7].
Nei prossimi paragrafi l'obbiettivo sarà quindi quello di spiegare, seppur ad alto livello, il funzionamento dei diversi classificatori.
Basandoci sull'interpretabilità dell'algoritmo da parte dell'umano, sarà possibile distinguere:

- ***Modello interpretabile:*** il modello di classificazione fornirà delle informazioni comprensibili che aiuteranno l'esperto di dominio a capire le ragioni delle decisioni intraprese dall'algoritmo.
  Un esempio di modello interpretabile è dato dagli Alberi di Decisione che basano le scelte su relazioni del tipo $X < Y$ (Figura 3.3):

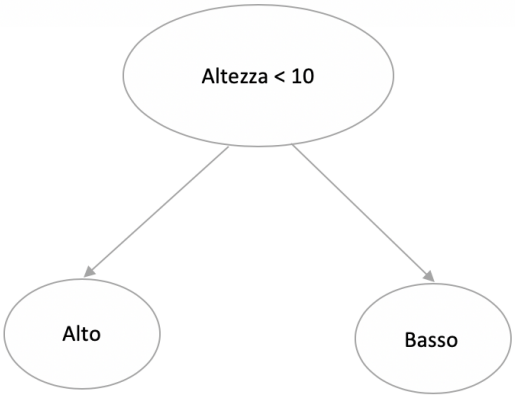


Figura 3.3: Esempio di criterio di decisione di un albero di decisione

Sarà facile capire che l'algoritmo deciderà se un item è alto o basso in base ad un'informazione riguardante la sua altezza.
Nel caso della Figura 3.3, se sarà vera la condizione $Altezza < 10$, verrà intrapresa la decisione di destra ($Basso$). In caso contrario verrà intrapresa la decisione di sinistra.

- ***Modello non interpretabile:*** il modello di classificazione non fornirà nessuna informazione sul tipo di decisione intrapresa.
  Un classico esempio di modello non interpretabile è quello generato dalle Reti Neurali o dalle SVM.

3.2.1 Gli Alberi di Decisione

Un albero di decisione può essere rappresentato da una struttura ad albero (Figura 3.4), nella quale:

- ***Nodo:*** contiene l'*attributo di split*, ovvero un attributo del training set utilizzato per intraprendere la decisione.
  Verrà spiegato in seguito come si sceglierà ad ogni step il nodo da utilizzare per lo split;

- ***Arco:*** viene definito come la connessione tra nodo padre e nodo figlio.
  Questo rappresenta il cuore della decisione, in quanto da ogni nodo possono essere intraprese scelte diverse secondo un certo criterio di split ben definito in fase di addestramento;

- ***Foglia:*** rappresenta l'etichetta di classe, ovvero la decisione finale dell'algoritmo.

Supponiamo di voler prevedere le condizioni meteo di domani basandoci sulle attuali condizioni meteorologiche.

A seguito della fase di addestramento, l'albero di decisione assumerà una struttura come la seguente:

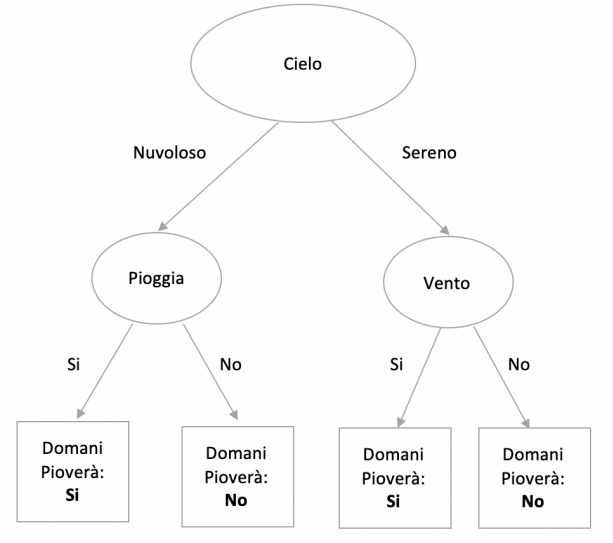


Figura 3.4: Albero di decisione

Immaginando che oggi sia una giornata in cui il cielo è sereno e non ci sia vento, intuiamo tramite il modello interpretabile della Figura 3.4 che domani probabilmente sarà una giornata senza pioggia.
In realtà, possiamo pensare la frase di cui sopra espressa attraverso un'entry di un dataset sprovvista di etichetta di classe, in cui *cielo* e *vento* sono due attributi:

Tabella 3.2: Entry che rappresenta le condizioni meteo di oggi

Cielo	**Vento**	**Pioggia**	**Domani Piove**
Sereno	No	No	?

In fase di classificazione, l'entry riportato nella Tabella 3.2 verrà immessa all'interno dell'albero di decisione a partire dal nodo padre e, per ogni livello dell'albero, verrà confrontato il valore dell'attributo con i valori delle possibili decisioni, così da prendere quella ritenuta corretta dall'algoritmo.
Sono importanti due considerazioni:

- Non è necessario che vengano valutati tutti gli attributi di un'entry.
  Nel nostro caso vengono presi in considerazione *cielo* e *vento*, ignorando così l'attributo *pioggia*.
  Il modello può essere un albero sbilanciato secondo il quale l'algoritmo ha ritenuto

che sia necessaria la valutazione di poche informazioni per stabilire l'etichetta di classe;

- Gli archi uscenti da un nodo devono rispondere a tutti i possibili valori che un attributo potrà assumere, sia nel caso di valori categorici che discreti.

L'ultima considerazione implica la possibilità di avere, per uno split, due strutture differenti (Figura 3.5):

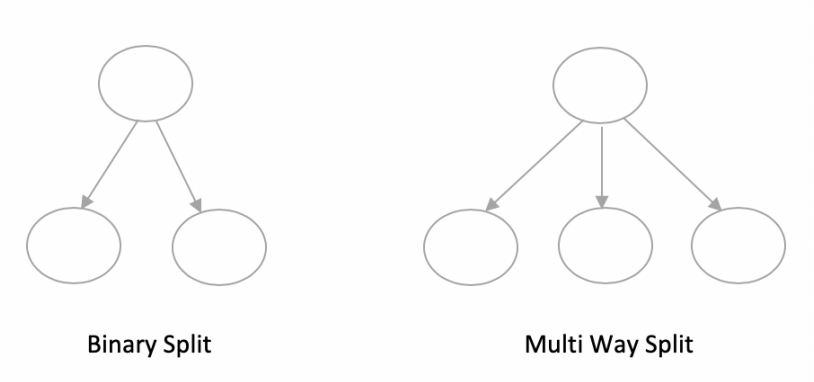


Figura 3.5: Possibili strutture di un nodo

Nel caso di una decisione ***Binary Split***, i valori ammissibili dell'attributo verranno divisi in due subset nel caso della classificazione categorica, altrimenti il criterio di split sarà nella forma $X < Y$ per la classificazione di valori discreti.
In una decisione del tipo ***Multi Way Split***, verranno generati più di due subset di valori ammissibili ed anche in questo caso ogni arco corrisponderà ad un subset.

3.2.2 Classificatore Bayesiano

Il classificatore Bayesiano basa il proprio funzionamento sull'utilizzo del teorema di Bayes.
In primo luogo verrà proposta una breve spiegazione del teorema di Bayes, successivamente verrà illustrata una descrizione ad alto livello dell'algoritmo di classificazione che sfrutta tale teorema.

3.2.2.1 Il teorema di Bayes

Definiamo la ***probabilità condizionata*** come la probabilità che si verifichi un evento A sapendo che si sia verificato l'evento B:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{3.1}$$

Dove definiamo:

- $P(A \cap B)$ come ***probabilità congiunta***, ovvero la probabilità che i due eventi si verifichino nello stesso momento;
- $P(B)$ come ***probabilità a priori***, ovvero la probabilità che si verifichi B, non tenendo conto di A.

Utilizzando la definizione di probabilità condizionata (3.1), possiamo scrivere:

$$P(A \cap B) = P(A|B)P(B) \tag{3.2}$$

$$P(B \cap A) = P(B|A)P(A) \tag{3.3}$$

Poiché $P(A \cap B) = P(B \cap A)$, sarà vera la relazione:

$$P(A|B)P(B) = P(B|A)P(A) \tag{3.4}$$

Dalla definizione (3.4) nasce il ***teorema di Bayes***, per cui:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{3.5}$$

Poniamoci ora nel caso della classificazione, dove A rappresenta l'etichetta di classe da voler predire e B la serie di coppie attributo-valore che compongono un certo dato di test.
Se è vera l'ipotesi d'indipendenza tra la serie di coppie attributo-valore, allora sarà vero che:

$$P(B|A) = \prod_{i=1}^{k} P(a_i|A) \tag{3.6}$$

dove k è il numero di attributi di un dato di test ed a_i è il valore dell'attributo *i-esimo*.
Tale ipotesi viene detta ***ipotesi Naïve***, la quale rappresenta un'assunzione molto forte poiché indica che la probabilità che un certo dato di test assuma l'etichetta di classe A potrà essere ottenuta da una semplice moltiplicazione, come nel caso della formula (3.7):

$$P(A|B) = \prod_{i=1}^{k} P(a_i|A)P(A) \tag{3.7}$$

Di seguito un esempio in cui viene presa in considerazione l'assunzione Naïve.
La Tabella 3.3 indica se, un certo apprendista Joda, dotato di determinate caratteristiche, farà parte o meno del Lato Oscuro [8]:

Tabella 3.3: Dataset apprendisti Lato Oscuro

Età	**Apprendistato completato**	**Disposizione**	**Specie**	**Lato Oscuro**
5	1	Felice	Umana	0
9	1	Felice	Gungan	0
6	0	Felice	Wookie	0
6	1	Triste	Calamari	0
7	0	Triste	Umana	0
8	1	Arrabbiata	Umana	0
5	1	Arrabbiata	Ewok	0
9	0	Felice	Ewok	1
8	0	Triste	Umana	1
8	0	Triste	Umana	1
6	0	Arrabbiata	Wookie	1
7	0	Arrabbiata	Calamari	1

Supponiamo di voler classificare il dato di test riportato nella Tabella 3.4, capendo se l'apprendista apparterrà o meno al Lato Oscuro:

Tabella 3.4: Dato di Test

Età	**Apprendistato completato**	**Disposizione**	**Specie**	**Lato Oscuro**
8	0	Arrabbiata	Umana	?

Il classificatore in primo luogo valuterà la probabilità a priori dei diversi valori assunti dall'etichetta di classe:

$$P(LatoOscuro = 1) = \frac{5}{12} = 0.416 \tag{3.8}$$

$$P(LatoOscuro = 0) = \frac{7}{12} = 0.483 \tag{3.9}$$

In secondo luogo verranno calcolate le diverse probabilità $P(a_i|A)$ sfruttando la condizione di indipendenza tra i vari attributi del dato di test:

$$P(Età = 8|LatoOscuro = 1) = \frac{2}{5} = 0.4 \tag{3.10}$$

$$P(Età = 8|LatoOscuro = 0) = \frac{1}{7} = 0.143 \tag{3.11}$$

$$P(ApprendistatoCompletato = 0|LatoOscuro = 1) = \frac{5}{5} = 1 \tag{3.12}$$

$$P(ApprendistatoCompletato = 0|LatoOscuro = 0) = \frac{2}{7} = 0.286 \tag{3.13}$$

$$P(Disposizione = Arrabbiato|LatoOscuro = 1) = \frac{2}{5} = 0.4 \tag{3.14}$$

$$P(Disposizione = Arrabbiato|LatoOscuro = 0) = \frac{2}{7} = 0.286 \tag{3.15}$$

$$P(Specie = Umana|LatoOscuro = 1) = \frac{2}{5} = 0.4 \tag{3.16}$$

$$P(Specie = Umana|LatoOscuro = 0) = \frac{3}{7} = 0.429 \tag{3.17}$$

Applicando l'equazione (3.7) alle espressioni [(3.8) - (3.17)] sarà possibile calcolare la probabilità che il dato di test appartenga o meno alla classe Lato Oscuro:

$$P(TestData|LatoOscuro = 1) = 0.416 \cdot 0.4 \cdot 1 \cdot 0.4 \cdot 0.4 = 0.006656 \tag{3.18}$$

$$P(TestData|LatoOscuro = 0) = 0.583 \cdot 0.143 \cdot 0.286 \cdot 0.286 \cdot 0.429 = 0.002925 \tag{3.19}$$

Poiché il valore dell'espressione (3.18) è maggiore rispetto a quello dell'espressione (3.19), il dato di test riportato nella Tabella 3.4 verrà etichettato come appartenente alla classe *Lato Oscuro = 1*, con probabilità:

$$P(TestData|LatoOscuro = 1) = \frac{0.006656}{0.006656 + 0.002925} = 0.69 \tag{3.20}$$

3.2.3 La classificazione basata su regole

Come detto precedentemente nella Sezione 3.1.4, le regole d'associazione sono regole nella forma

$$R : X \implies Y$$

dove X è detto *antecedente (o corpo)* ed Y è detto *conseguente (o testa)* della regola.
Lo scopo della classificazione basata su regole è quello di costruire un classificatore dove il modello è fondato sulle regole estratte da un set di dati.
Capiamo che nel nostro caso, la definizione di regola dovrà essere vista come segue:

$$TR : D \implies C \tag{3.21}$$

dove:

- D: rappresenta il dato di training o test, composto da diversi item d del tipo *attributo=valore*;
- C: rappresenta l'etichetta di classe.
  È importate notare che non sarà garantita la presenza dell'etichetta di classe (e quindi della testa della regola), in quanto essa non è conosciuta per un dato di test.

Supponiamo di voler classificare un dato di test TD e di essere a disposizione di n regole di training nella forma (3.21). Una regola di training TR verrà presa in considerazione

per la classificazione del dato di test TD se sarà verificata la condizione:

$$TD \subseteq TR \tag{3.22}$$

ovvero TR sarà contenuta dal nostro dato di test TD.
A titolo d'esempio, si definisce TD come riportato nella Tabella 3.5:

Tabella 3.5: Dato di Test

Peso	**Altezza**	**Condizione Fisica**
Range1	Range2	?

dove le etichette *Range-n* rappresentano dei valori precedentemente discretizzati grazie ad un'antecedente fase di preprocessing.
Supponiamo che, a seguito dell'utilizzo di un algoritmo d'estrazione di regole, vengano estratte le seguenti regole d'associazione:

$$Peso = Range3, Altezza = Range2 \implies Sovrappeso \tag{3.23}$$

$$Peso = Range1, Altezza = Range1 \implies Sottopeso \tag{3.24}$$

$$Peso = Range1, Altezza = Range2 \implies Sottopeso \tag{3.25}$$

Come da definizione (3.22), verrà scelta la regola (3.25), così che il dato di test verrà etichettato con la classe *Sottopeso*.

3.2.4 Support Vector Machines

Ipotizziamo di voler classificare un dataset formato da n attributi.
Una Support Vector Machines (SVM) rappresenterà tutte le entry del dataset in un piano formato da un numero di assi pari al numero di attributi del dataset stesso.
In questo capitolo, per semplicità, verrà trattato il problema come bidimensionale.
In realtà, dopo averne capito il funzionamento, l'algoritmo sarà estendibile su più dimensioni.
In aggiunta a quanto visto finora, le SVM hanno l'importante proprietà di poter risolvere anche problemi di regressione, ovvero classificazioni il cui valore in output sarà un valore continuo (nel prosieguo di questa tesi non verranno trattati problemi di regressione).

3.2.4.1 Funzionamento di una SVM

Le SVM costruiscono uno spazio formato da n piani, dove n rappresenta il numero di attributi della collezione di dati.
Ogni entry di tale collezione, occuperà quindi una posizione ben precisa nello spazio, dipendente dal valore degli attributi caratterizzanti.
Il classificatore costruirà, intorno a tali punti, delle *rette* o *iperpiani* (a seconda del

modello bidimensionale o n-dimensionale) utili ai fini della classificazione.
In una SVM si possono distinguere tre caratteristiche principali:

- ***Iperpiano:*** rappresenta la separazione tra le varie classi.
  Sarà proprio grazie agli iperpiani costruiti dall'algoritmo che il modello discriminerà un elemento come appartenente alla classe A o B, a seconda della posizione che esso occupa rispetto all'iperpiano di separazione;

- ***Margine:*** rappresenta la distanza minima tra elementi di classe diversa.
  Il margine viene utilizzato come misura per determinare il miglior iperpiano di separazione;

- ***Vettore di supporto:*** rappresenta un particolare dato di training.
  In generale i dati possono essere visti come vettori, dove l'elemento x_i rappresenta il valore dell'attributo *i-esimo*.
  Nel caso in cui tali dati partecipino attivamente alla definizione del margine, vengono detti vettori di supporto.

Poniamoci ora nel caso in cui si voglia classificare una collezione di dati dotata solamente di due attributi più l'etichetta di classe, così da poter lavorare in uno spazio bidimensionale.
Definiamo *Kernel* di una SVM la funzione che agisce come muro di separazione tra gli elementi di classi diverse.
A seconda della disposizione degli attributi su di un piano, potranno essere distinti due casi:

- ***SVM Lineare:*** i dati rappresentati nella Figura 3.6 risultano separabili da una funzione di tipo lineare.

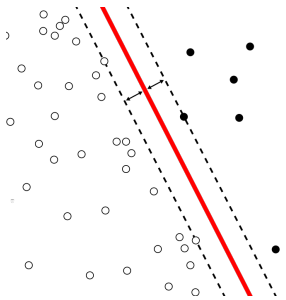


Figura 3.6: SVM lineare [9]

Come si può evincere dalla figura, si possono trovare numerose rette capaci di discriminare le due classi.
Ma come selezionare la migliore?
La migliore retta di separazione sarà quella che minimizza l'errore di classificazione.

Per far ciò sarà necessario scegliere l'iperpiano che massimizza il margine tra le due classi:

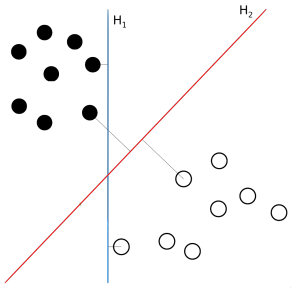


Figura 3.7: Scelta della migliore funzione di separazione [9]

Nella Figura 3.7, si vede chiaramente che il margine m_2 della retta H_2 è più grande rispetto al margine m_1 della retta H_1.
Per tal motivo l'algoritmo sceglierà H_2 come retta di separazione.

- ***SVM Non-Lineare:*** nella Figura 3.8 i dati risultano separabili solamente tramite l'utilizzo di una funzione di tipo non lineare.

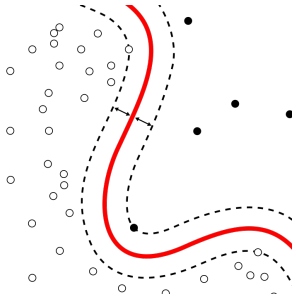


Figura 3.8: SVM non lineare [9]

In tal caso non sarà possibile scegliere la funzione di separazione con la strategia esposta precedentemente, poiché non possiamo utilizzare una funzione di separazione lineare.
Ad alto livello, l'algoritmo seguirà due step principali:

1. I dati vengono resi linearmente separabili, a seguito di una conversione verso un altro spazio di rappresentazione;
2. Viene applicata la strategia delle SVM con Kernel lineare.

3.2.5 K-Nearest Neighbours

I classificatori spiegati finora costruiscono internamente un modello (e.g. SVM costruisce degli iperpiani di separazione) che permette di assegnare gli attributi di classe ai dati non classificati.
Appartiene alla categoria degli *instance based classifiers* il classificatore **K-Nearest Neighbours (K-NN)**.
Con *instance based classifier* intendiamo un tipo di classificatore che utilizza i dati di training per classificare i dati non etichettati.
Possiamo distinguere:

- Rote-learner: classificatori che memorizzano l'intera collezione di dati e, al momento della classificazione, ricercano un match tra *collezione di dati - dato da classificare*;
- Nearest neighbor: si utilizzano solamente i K punti più vicini al dato training per etichettare il dato incognito.

K è un parametro intero scelto dall'utente che utilizza il classificatore e rappresenta il numero di vicini da considerare al momento della classificazione.

Immaginiamo di voler capire se il cerchio verde X della Figura 3.9 appartenga alla classe *quadrato blu* o *triangolo rosso*:

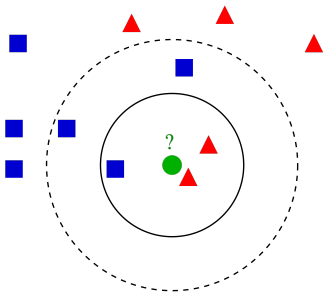


Figura 3.9: K-NN [10]

L'algoritmo seguirà i seguenti passi:

1. Viene calcolata la distanza tra X e gli altri dati del training set. Per misurare la distanza tra due punti p e q nello spazio, viene utilizzata la definizione di *distanza euclidea*:
$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2} \tag{3.26}$$
2. Vengono identificati i K punti più vicini;

3. Viene utilizzata la classe dei K vicini (voto di maggioranza) per determinare la classe del dato incognito X.

Questo sistema chiaramente richiede la rappresentazione di tutti i punti del training set all'interno di uno spazio *n-dimensionale*, dove n rappresenta il numero di attributi della collezione di dati.

Dalla Figura 3.9 si può evincere l'importanza del parametro K:

- Se $K = 3$, il dato X verrà etichettato come *triangolo rosso*, in quanto all'interno del cerchio non tratteggiato sono presenti due *triangoli rossi* e solamente un *quadrato blu*;
- Se $K = 5$, il dato X verrà etichettato come *quadrato blu*, in quanto all'interno del cerchio tratteggiato sono presenti tre *quadrati blu* e due *triangoli rossi*;

All'utilizzo di questo algoritmo, dovrà precedere una fase di scelta del parametro K.
Scegliere un valore K troppo basso significa esporre la classificazione a punti di rumore, d'altro canto un valore troppo alto significa includere nel processo di classificazione punti molto lontani dall'incognita X, che potenzialmente apparterranno ad un'altra classe.

3.2.6 Le Reti Neurali

Le *Reti Neurali* (*NN* dall'inglese *Neural Networks*) si ispirano alla rete neurale biologica, in quanto le unità fondamentali che compongono il modello sono il *neurone* e le *connessioni* (come se fossero sinapsi) che mettono in contatto i vari neuroni.
Passando ad una definizione più formale, una NN può esser definita come un modello matematico costituito da diverse unità d'elaborazione interconnesse tra loro.
L'unità d'elaborazione viene chiamata neurone u_i (o nodo) e il suo unico scopo consiste nel prendere in input una serie di valori e rilasciare in output un unico valore, calcolato grazie ad una certa funzione.
Possono esser distinte, in base al ruolo che ricoprono, tre tipi di neuroni:

- Neurone di input: riceve in ingresso gli input del modello;
- Neurone di output: fornisce in output la risposta del modello;
- Neurone di Hidden: collega tra loro diversi neuroni, giacendo su un livello intermedio.

La Figura 3.10 mostra la composizione di una NN.

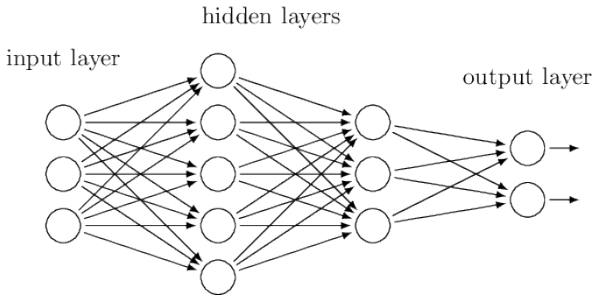


Figura 3.10: Struttura di una rete neurale

Un nodo *i-esimo* viene connesso al neurone successivo *j-esimo* grazie ad una connessione w_{ij}.
Quest'ultimo parametro viene definito come *peso della connessione* ed in base al suo valore è possibile distinguere:

- $w_{ij} > 0$: connessione eccitatoria;
- $w_{ij} < 0$: connessione inibitoria;
- $w_{ij} = 0$: nessuna connessione.

I valori w_{ij} sono fondamentali per la creazione del modello, in quanto possono esser pensati come la fonte di conoscenza della rete stessa.
È importante precisare che la memoria non nasce programmaticamente, ma viene acquisita attraverso una *fase di apprendimento*.
Tale fase tarerà la rete correttamente e stabilirà i pesi delle connessioni.
Ogni nodo u_i è caratterizzato da un certo valore, detto *valore di attivazione* $a_i(t)$, dove per t si intende il momento in cui viene eseguito il calcolo.
Il valore di attivazione è molto importante, poiché è parte integrante dell'output o_i del neurone u_i.
L'output del neurone dipenderà dal valore corrente a_i, trasformato attraverso una funzione di output:

$$f_i(a_i(t))$$

La funzione f_i dipende fortemente dall'implementazione utilizzata: può essere una semplice funzione $f(x) = x$, oppure una funzione soglia, che restituisce un output pari o maggiore di 0 a seconda del valore in input, così da influenzare o meno le altre unità connesse.

Definita la struttura principale della rete, rimane da capire come valutare il valore $a_i(t)$ e come addestrare la rete.
Immaginando che un nodo sia dotato di diverse connessioni, il valore in input per esso si può pensare nel caso più semplice come la somma di tutti gli output del livello

precedente:

$$net_i = \sum_j w_{ji} o_j$$

Tale misura di input, risulta fondamentale nel calcolo del valore di attivazione.
La funzione di attivazione F è una funzione che riceve come parametri il valore corrente di attivazione $a_i(t)$ e di input net_i e produce un nuovo valore di attivazione al tempo $t+1$:

$$a(t+1) = F(a(t), net_i(t))$$

Questa funzione gioca un ruolo fondamentale nella qualità della rete, in quanto influenza fortemente il valore di output.
Funzioni di attivazione diverse comportano risultati notevolmente diversi, anche con una stessa struttura della rete.
Quelle più comuni sono le seguenti:

- Funzione Soglia

$$\begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

- Funzione Sigmoide

$$y = (1 + e^{-z})^{-1}$$

- Funzione Tangente Iperbolica

$$y = tanh(z)$$

In ultimo, si vuol chiarire come possono essere definiti i pesi w_{ij} che determinano le connessioni della rete.
In questa sezione non si vuole entrare nel dettaglio matematico ma si vorrà dare solamente una spiegazione ad alto livello della fase di training.
Tra i vari tipi di apprendimento possibili, in questa tesi verrà utilizzata la *backpropagation*.
Com'è intuibile dal nome, la fase di *backpropagation* consiste in una modifica dei pesi della NN calcolando l'errore sul livello successivo così da aggiornare il peso precedente.
In particolare, la variazione sul peso che dovrà esser applicata ad una connessione è data dalla seguente formula:

$$\Delta_p w_{ji} = \eta \delta_{pi} o_{pj}$$

dove:

- Δ_p: valore della modifica da applicare sul peso;
- η: tasso di apprendimento ($0 < \eta < 1$);
- δ_{pi}: errore compiuto dall'unità d'arrivo (previsto - attuale);
- o_{pj}: valore dell'output dell'unità di partenza.

Queste informazioni di base, costituiscono il cuore di una Rete Neurale.
Nello specifico, di nostro interesse è lo studio delle Reti *MLP* (*Multi Layer Perceptron*).
Il *Perceptron* è definito come la versione più semplice di una NN, ovvero una rete composta da solamente un neurone che agisce sia da input che da output.
Estendendo tale considerazione, si ha la nascita del Multi Layer Perceptron, dove diversi percettroni vengono combinati così da compartecipare alla formazione degli strati di input, output e hidden.

3.2.7 Considerazioni sulle Reti Neurali

Si può immaginare che, a seconda del numero di layer utilizzati, la NN corrisponda alla creazione di un modello matematico differente nello spazio di rappresentazione.
Per spiegare questo concetto si farà riferimento alla Figura 3.11.
In essa vengono riportati, oltre al modello di NN MLP utilizzato, due casi di studio differenti:

- Problema dello XOR: nello spazio vengono rappresentati i punti relativi all'OR Esclusivo (0,0), (0,1), (1,0), (1,1).
- Regioni generiche: vengono rappresentate delle regioni convesse nello spazio.

In entrambi i casi, l'obiettivo sarà trovare una retta o un iperpiano di separazione che ci permetta di dividere agevolmente l'area appartenente alla classe A e alla classe B.
Sia nel caso dello XOR che nel caso delle regioni convesse vengono indicate le porzioni di spazio appartenenti alle due classi.
A seconda del numero di layer, possono verificarsi le seguenti possibilità:

- *1 layer*: la rete corrisponde al singolo percettrone che riceve un input e dà in risposta un valore di output.
  In questo caso verrà creata una singola retta di separazione, dove il coefficiente angolare m e l'ordinata all'origine q saranno valorizzati dai coefficienti della rete neurale.
  Com'è visibile nella figura, con un solo livello non sarà possibile risolvere il problema proposto, ma sarà comunque implementabile una retta di separazione che permetta di separare due classi ben distinte nello spazio;
- *2 layers*: si innalza la complessità della rete.
  In questo caso potranno essere tracciate due rette di separazione che consentono di classificare correttamente i dati relativi al caso dello XOR.
  Di difficile risoluzione rimane la classificazione di regioni convesse;
- *3 layers*: si ha un incremento notevole della complessità del modello, permettendo la risoluzione dei due problemi proposti.
  In questo caso, la rete sarà capace di creare delle regioni di separazione complesse capaci di separare agevolmente le classi proposte.

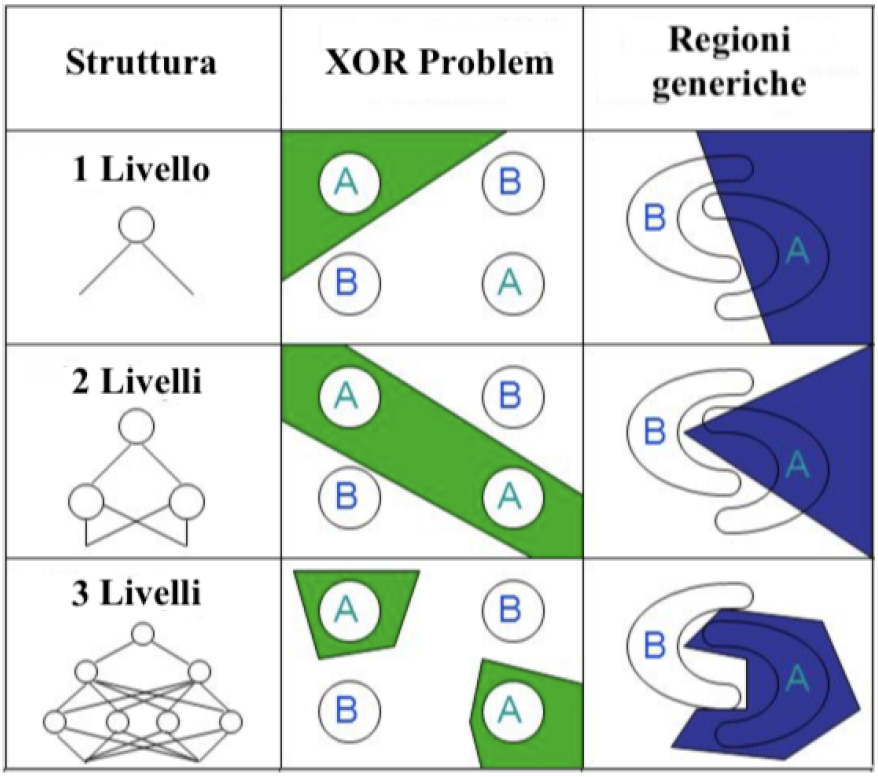


Figura 3.11: Modello matematico di una Rete Neurale

Queste considerazioni sulla base matematica di una NN (talvolta trascurate) sono di fondamentale importanza.
Nella fattispecie, viene rilevata una forte analogia con il modello SVM, data dalla creazione di rette nello spazio (anche se secondo diversi principi di training) che consentono la separazione di classi.

3.3 Classificazione basata su Bagging e Ensemble

In certi casi è possibile aumentare le prestazioni di un algoritmo di classificazione utilizzando le tecniche di ***Bagging*** ed ***Ensemble***.
Che si parli della prima o della seconda tecnica, entrambe si basano sul concetto di *voto di maggioranza*: inizialmente vengono addestrati, con una certa strategia, diverse versioni del medesimo algoritmo. Successivamente, verrà classificato il dato di test con ogni modello e verrà assegnata l'etichetta di classe più predetta.
In base al tipo di strategia adottata, si possono distinguere le seguenti tecniche:

- ***Bagging:*** lavora a livello di dato.
  Vengono generate n versioni differenti del training set effettuando un campionamento casuale con ripetizione partendo dai dati del training set originale.

Per ogni training set n_i, viene generato un modello differente che verrà utilizzato per la classificazione.

- ***Ensemble:*** lavora a livello d'algoritmo.
  A differenza del Bagging, i dati di training saranno i medesimi per ogni modello, quello che cambierà sarà la configurazione del modello stesso.

3.3.1 Random Forest

Il ***Random Forest*** è una variante degli *Alberi di Decisione* che basa la propria strategia sull'utilizzo di tecniche di *Bagging* ed *Ensemble*.
Esso opera costruendo una serie di Alberi di Decisione e, seguendo la strategia *Ensemble* vista precedentemente, restituisce in output l'etichetta di classe maggiormente votata dalla serie di alberi.
Gli n Alberi di Decisione vengono costruiti con dataset differenti, dove ognuno di essi è creato sfruttando la tecnica del Bagging.
Oltre ad avere un dataset di training differente, cosa distinguerà un albero da un altro? Dobbiamo aspettarci che i risultati prodotti dagli n classificatori non siano correlati tra di loro, altrimenti non avrebbe senso utilizzare alberi diversi ma che elaborerebbero gli stessi risultati.
Per far ciò, degli m attributi di split selezionabili dall'albero di decisione, ne verrà scelto solamente un sottoinsieme.
Inoltre, ogni albero baserà la propria scelta, ad ogni livello, su un attributo differente dagli altri alberi.
Se non fosse così, tutti gli alberi sarebbero costruiti allo stesso modo, ovvero partendo dall'attributo con maggiore importanza, con una conseguente riduzione della variabilità del modello.

3.3.2 Gradient Boosting

Il termine *Boosting* è associato all'idea di miglioramento: un classificatore, di per sè debole, attraverso l'utilizzo di apposite metodologie può esser migliorato così da ottenere prestazioni migliori.
Alla base, vi è l'intento di costruire diversi classificatori in modo sequenziale, dove la differenza tra i vari modelli è data dal training set.
La sequenza con cui vengono costruiti i modelli d'apprendimento può esser definita adattiva, in quanto ad ogni passo si cercano di pesare in maniera differente i dati che causano misclassificazione in modo tale da tenerne conto al passo successivo.
Così sarà possibile costruire diversi modelli predittivi dove di volta in volta si pone più enfasi ai dati che il classificatore non è in grado di predire.
Tra i diversi algoritmi di Boosting, il *Gradient Boosting* ad ogni passo utilizza la discesa del gradiente per la costruzione dei vari training set.

4 Il processo di classificazione

Finora è stato spiegato il processo di *Data Mining* e quali sono gli algoritmi noti in letteratura che possono essere sfruttati per effettuare la classificazione di un dato incognito. In questa sezione verranno spiegate le implementazioni di strategie che combinano tra loro diversi algoritmi di classificazione e clustering al fine di ottenere il miglior livello qualitativo possibile nella predizione di una collezione di dati.
In questo senso, si agirà nel tentativo di classificare una particolare tipologia di dataset, definita come *Imbalanced Dataset* (Sezione 4.2).
I linguaggi di programmazione utilizzati per l'implementazione degli algoritmi presentati di seguito saranno Python e Java.
Per il primo si sfrutterà maggiormente la libreria di Machine Learning Scikit-learn [11], invece per il secondo si utilizzeranno gli algoritmi d'estrazione e classificazione basati su regole d'associazione.

4.1 Scikit Learn

Scikit Learn [11] è una libreria open source disponibile per Python.
Essa mette a disposizione le implementazioni di diversi algoritmi di:

- ***Classificazione:*** utile per l'assegnazione dell'etichetta di classe a dati non ancora classificati.
  Vengono qui implementati gli algoritmi di classificazione basati su alberi di decisione, teorema di bayes, SVM, K-NN, Random Forest ed altri ancora visti nelle sezioni del capitolo 3;

- ***Regressione:*** per l'assegnazione di un valore continuo ad un attributo di classe;

- ***Clustering:*** per il raggruppamento di dati con alta similarità all'interno dello stesso cluster.
  Vengono qui implementati il K-Means e il DBSCAN visti rispettivamente nelle sezioni 3.1.2 e 3.1.3.

- ***Dimensionality reduction:*** per la riduzione del numero di features (e.g. PCA, LDA) utilizzate durante la classificazione, permettendo così di prendere in considerazione le features con più alta rilevanza;

- ***Model Selection:*** per la valutazione delle prestazioni degli algoritmi di classificazione e clustering.
  Grazie a tali algoritmi possiamo gestire in modo semplice Holdout, Cross Validation, Leave One Out, visualizzazione della matrice di confusione, Precisione Richiamo e F1-Score;

- ***Preprocessing:*** utile per la fase di preprocessing che precede la classificazione o il cluster.
  Vengono qui implementati algoritmi di discretizzazione e normalizzazione.

4.2 Dataset Sbilanciati

Supponiamo di avere un dataset per il quale ad ogni entry è associata la classe A o B. Per **dataset sbilanciato** si intende una particolare collezione di dati per cui il numero di entry appartenenti alla classe A differisce dal numero di entry della classe B.
Tale problematica trova riscontro in numerosi domini, ad esempio:

- In ambito medico, dove il numero di pazienti classificati come portatori di una certa malattia è di molto minore rispetto al numero di pazienti sani;
- Nell'analisi di e-mail che devono essere classificate come ham o spam: il numero di e-mail classificate come spam è di gran lunga minore rispetto all'ham;
- Nell'ambito assicurativo, dove solamente una piccola parte degli assicurati hanno causato un incidente.

Per comprendere quanto un dataset sia effettivamente sbilanciato, viene utilizzata la misura *Imbalanced Ratio (IR)*:

$$IR = \frac{\#MajClass}{\#MinClass} \tag{4.1}$$

dove *#MajClass* indica il numero di elementi della classe maggioritaria, mentre *#MinClass* indica il numero di elementi della classe minoritaria.
Sarà lecito pensare che:

- Per $IR \simeq 1$ non si riscontrano particolari problematiche data la numerosità della classe maggioritaria circa uguale al numero di elementi della classe minoritaria;
- Per IR molto più grandi del valore 1, si incominceranno ad avere difficoltà nella classificazione.

Per ognuno dei dataset di prova utilizzati d'ora in avanti, verrà specificato qual è il grado di IR.
Per effettuare l'analisi sperimentale saranno impiegati i dataset di Benchmark UCI [12].

4.3 La gestione dei dataset sbilanciati allo stato dell'arte

La gestione dei dataset sbilanciati è una problematica già affrontata in letteratura e che trova diverse soluzioni.
Lo scopo di questa tesi in primo luogo sarà quello di sfruttare queste soluzioni già note, ma anche di utilizzarle a proprio favore per creare degli algoritmi che ne migliorino i limiti.
Le tecniche di *Oversampling* e *Undersampling* lavorano a livello di dato (*Data Level Approach*) tentando di rimodellare, con delle strategie che verranno spiegate in seguito, la collezione di dati in modo tale da portare il fattore IR il più vicino possibile a 1.
Le tecniche dell'Oversampling e dell'Undersampling seguono due approcci opposti.
Supponiamo di avere un dataset formato da 10000 e-mail classificate come *ham* ed altre 500 come *spam*:

- l'Oversampling agirà sulla classe minoritaria (spam) creando, con una certa metodologia, altre entry etichettate come spam;
- l'Undersampling agirà sulla classe maggioritaria (ham), eliminando delle entry ritenute poco rilevanti.

Nel seguito verranno analizzate nel dettaglio queste due tecniche, cercandone di capire vantaggi e svantaggi e come si possa intervenire sui punti di debolezza per aumentarne il rendimento.

4.3.1 L'Oversampling

Come anticipato nella Sezione 4.3, l'Oversampling consiste nell'aumentare il numero di entry appartenenti alla classe minoritaria, così da evitare il fenomeno dell'overfitting verso quella maggioritaria.
Ai fini dell'implementazione possono essere seguite due strade differenti:

- ***Random Oversampling:*** i dati della classe minoritaria vengono replicati in maniera casuale.
  Questa metodologia è di facile implementazione ed estremamente veloce durante la fase d'esecuzione in quanto consiste nella sola replica del dato.
  Tuttavia potrebbe essere poco performante poiché la replica non viene fatta in maniera intelligente, rischiando così di escludere dall'Oversampling informazioni della classe minoritaria molto rilevanti ma poco presenti.
  Si tenta di superare tale limite sfruttando lo SMOTE Oversampling.

- ***SMOTE Oversampling***[13]: è l'acronimo di *Synthetic Minority Over-sampling TEchnique*. Questo algoritmo crea, partendo dai dati e dalle features originali, nuove entry non presenti all'interno del dataset.
  Immaginiamo di avere un dataset formato da due features e due etichette di classe: i fiori *setosa* e *versicolor* sono identificati dall'altezza e dall'ampiezza del proprio sepalo.
  Il dataset è così distribuito: 13 entry della classe *versicolor* (cerchi verdi) e 4 elementi della classe *setosa* (cerchi bordeaux). Per creare i nuovi dati, l'algoritmo parte dalle entry della classe minoritaria già esistenti generando, nelle linee di congiunzione presenti nella Figura 4.1, delle entry appartenenti alla classe *setosa* (cerchio rosso).
  Nella Figura 4.1 potenzialmente possono essere generati 8 nuovi elementi.

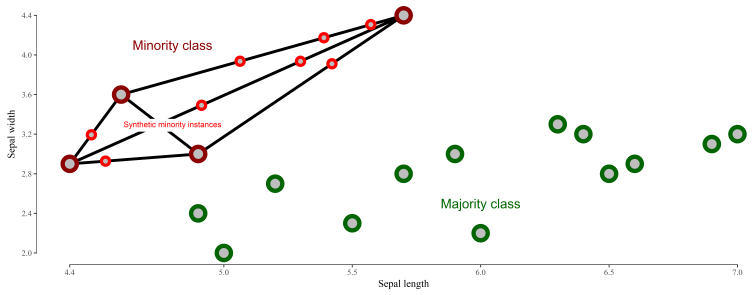


Figura 4.1: Funzionamento dello SMOTE Oversampling

Come vengono scelti gli elementi da generare?
Supponiamo di voler creare $dup\_size = 100$ elementi, dove dup_size è un numero intero che potrà essere passato come parametro alla funzione che implementa lo SMOTE.
L'algoritmo seleziona, partendo da un elemento casuale p della classe minoritaria, K vicini q_i per i quali verrà tracciata una linea d'unione $p - q_i$ (Figura 4.2).

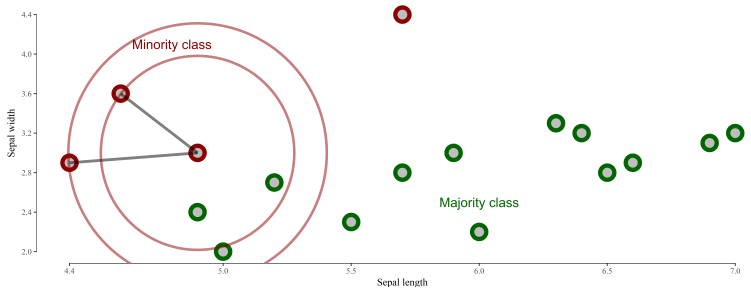


Figura 4.2: SMOTE per $K = 2$

Come secondo step verrà scelta una delle due line $p - q_i$ generate e verrà sintetizzato su di essa un nuovo elemento.
Il procedimento fin qui spiegato verrà ripetuto più volte, finché non saranno prodotti un numero di elementi pari a dup_size.

Nella Sezione 4.4 verrà presentata una terza strategia di Oversampling basata su clustering.

4.3.2 L'Undersampling

L'Undersampling si traduce in un approccio totalmente opposto rispetto all'Oversampling.

Esso consiste nell'eliminare dal dataset di training una parte delle entry appartenenti alla classe maggioritaria, così da ridurne la numerosità.
Come visto per l'Oversampling (vedi Sezione 4.3.1), tale approccio tende a ridurre l'overfitting verso la classe maggioritaria, portando il fattore IR verso il valore 1.
Il *Random Undersampling* consiste nell'eliminare, in modo casuale, dei dati appartenenti alla classe maggioritaria.
L'intero *del_size* indica la percentuale di elementi che dovranno essere eliminati dalla classe maggioritaria (e.g. $del\_size = 50$ indica che dovranno essere mantenuti solamente la metà degli elementi della classe maggioritaria).
Anche se molto efficiente dal punto di vista del tempo d'esecuzione, questo meccanismo potrebbe da una parte eliminare delle informazioni utili, mentre dall'altra, potrebbe lasciare all'interno informazioni della classe maggioritaria che sono già ben presenti nel training set.
Al fine di superare questa problematica, nella Sezione 4.4 verrà proposta una soluzione tramite la quale si evita l'eliminazione di dati significativi.

4.3.3 Validazione di dataset su cui è stato effettuato il sampling

Un'operazione di sampling modifica fortemente la quantità di dati all'interno del training set.
Basandoci sull'ultima considerazione, può essere opportuno adottare una ben precisa strategia di validazione del modello tra quelle viste nella Sezione 2.2.1.
Immaginiamo di avere a disposizione una collezione di dati così formata:

- 1000 entry appartenenti alla classe A (classe maggioritaria);
- 100 entry appartenenti alla classe B (classe minoritaria);
- $IR = \frac{1000}{100} = 10$.

Nell'utilizzo dell'Oversampling con $dup\_size = 500$, in totale si disporrà di un dataset finale dotato di 1600 elementi (1000 appartenenti alla classe maggioritaria e 600 alla classe minoritaria).
In tal caso si potrà decidere se applicare l'Holdout o la Cross Validation, in quanto disponiamo di una collezione di dati abbastanza ampia.
Scegliendo ad esempio la Cross Validation con $cv = 5$, si otterrà un training set di 1280 entry e un test set di 320 entry ad ogni iterazione.
In questa circostanza sarà molto dispendiosa l'applicazione della Leave One Out Validation, in quanto implicherebbe effettuare la fase di training 1600 volte.

Nell'ipotesi dell'Undersampling con $del\_size = 50$ si disporrà, a seguito dell'applicazione dell'algoritmo, di un dataset dotato di 600 elementi (500 appartenenti alla classe maggioritaria e 100 alla classe minoritaria).
In tal caso sarà obbligata la scelta della Leave One Out Validation.
Immaginiamo di voler applicare la Cross Validation con $cv = 5$, questo implicherà la

costruzione un training set di 480 elementi ed un test set di 120 entry ad ogni iterazione. Tuttavia, una tale dimensione per il training set risulta troppo ridotta per effettuare l'addestramento di un modello.
Per tal motivo sarà obbligata la scelta della Leave One Out Validation.
In letteratura viene consigliata l'applicazione:

- della Leave One Out Validation nel caso di training set con dimensione minore di 1000 entry;
- dell'Holdout o della Cross Validation in caso contrario.

Nel corso di questa tesi, per i processi di validazione del software predittivo sviluppato, si farà riferimento a quanto precedentemente esposto in questa sezione.

4.3.4 Classificazione basata su voto di maggioranza

Con *Algorithm Level Approach* si intende il tentativo di migliorare le performance del processo di classificazione agendo sulla strategia utilizzata mantenendo invariata la struttura e la dimensione del dataset di training.
Nelle sezioni successive verranno proposte, con i loro vantaggi e svantaggi, una serie di strategie che combinano algoritmi di classificazione e di clustering (come fatto per il sampling) al fine di capire come classificare al meglio il dato ignoto.
È importante precisare che il miglior risultato possibile si ottiene da una combinazione degli approcci esposti in questa sezione con le strategie di sampling spiegate precedentemente (vedi Sezione 4.3): non agire sulla collezione di dati, anche avendo a disposizione il miglior classificatore implementabile, significherà portarsi dietro tutti i limiti visti finora dati dalla numerosità della classe maggioritaria rispetto a quella minoritaria.
Scendendo ad un livello di dettaglio maggiore, nella Sezione 3.3 si è già parlato di combinare insieme più classificatori per migliorare le prestazioni:

- ***Bagging:*** utilizza diversi training set (creati a partire dal training set originale) per addestrare il medesimo classificatore;
- ***Ensemble*** utilizza lo stesso training set per addestrare il medesimo classificatore ma con configurazioni differenti.

Entrambi basano la propria decisione su un meccanismo a voto di maggioranza: la classe predetta maggiormente dalla serie di classificatori sarà la classe con cui verrà etichettato il dato di test. Come si può notare, entrambe le strategie hanno un punto in comune: utilizzare lo stesso classificatore.

Un approccio differente, come quello di cui ci si è avvalsi in questa sezione, si basa sull'utilizzo di classificatori diversi, su cui viene adoperato lo stesso training set:

1. Vengono scelti n classificatori differenti, dove n dev'essere un numero dispari;
2. Ogni classificatore *n-esimo* viene addestrato con il training set a disposizione;

3. Si classifica il dato di test *i-esimo* con ognuno dei classificatori a disposizione;

4. Viene assegnato al dato *i-esimo* l'etichetta di classe maggiormente predetta.

In tal modo sarà possibile correggere gli errori che potenzialmente sarebbero stati fatti da un classificatore che lavora singolarmente.

Un'alternativa alla strategia di scelta esposta nel punto 4 è l'utilizzo di un approccio *Cost Sensitive*.
Cosa si intende per approccio *Cost Sensitive*?
Immaginiamo di disporre di 3 algoritmi di classificazione (e.g. Decision Tree, SVM, Gaussian Classifier), dove ciascuno è stato precedentemente validato con il dataset a nostra disposizione.
Sapendo a priori quali sono le performance di ogni modello, sarà possibile assegnare ad ognuno di essi un punteggio (peso) che indica quanto sia "pesante" il voto del classificatore.
Il dato di test verrà così etichettato con l'etichetta di classe che ha ottenuto un punteggio maggiore.

4.3.5 Classificazione basata su Clustering

Gli algoritmi di clustering possono essere una grande fonte d'aiuto per associare il dato di test al miglior tipo di classificatore disponibile.
Nella Sezione 4.3.4 sono stati combinati insieme diversi classificatori per migliorare le performance del processo di classificazione, tuttavia non si è preso in considerazione il dato di test sottoposto alla classificazione.
La seguente strategia si basa sull'idea di associare al singolo, o ad un gruppo di dati di test, il miglior modello *i-esimo* disponibile, dove ogni modello verrà costruito su ognuno dei cluster prodotti dal processo di clustering.
Supponiamo di disporre di un dataset sbilanciato sul quale viene applicata la Holdout Validation cosicché, a seguito dell'applicazione della stessa, si disponga di un training set e di un test set.
Vengono proposte due strategie, in base a come viene trattato il test set (vedi Sezioni 4.3.6 e 4.3.7), le quali hanno in comune il modo in cui viene manipolato il training set.
Per capire come gestire quest'ultimo, facciamo riferimento alla Figura 4.3:

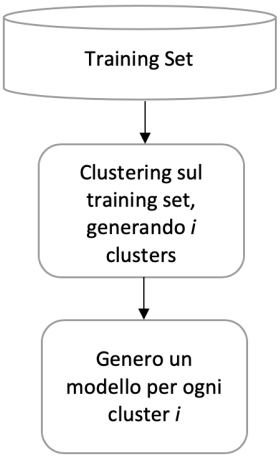


Figura 4.3: Clustered Training

1. Si applica il K-Means o il DBSCAN al training set.
   L'algoritmo selezionato genera i cluster differenti.
   Il numero di cluster generati può essere un numero a scelta dell'utente nel caso del K-Means oppure un numero incognito dato dall'algoritmo come nel caso del DBSCAN.

2. Si esegue la fase di training per ogni cluster *i-esimo*, generando così n modelli diversi, con $n = i$.

3. Viene applicata una delle strategie proposte nelle Sezioni 4.3.6 e 4.3.7.

Le due strategie differiscono per l'applicazione (o meno) del Clustering anche nel test set.
Segue una spiegazione ad alto livello nelle sezioni successive.

4.3.6 Classificazione basata su singolo dato di test

In questa metodologia di classificazione il training set rimane invariato.
Facciamo riferimento alla Figura 4.4 per spiegarne il funzionamento:

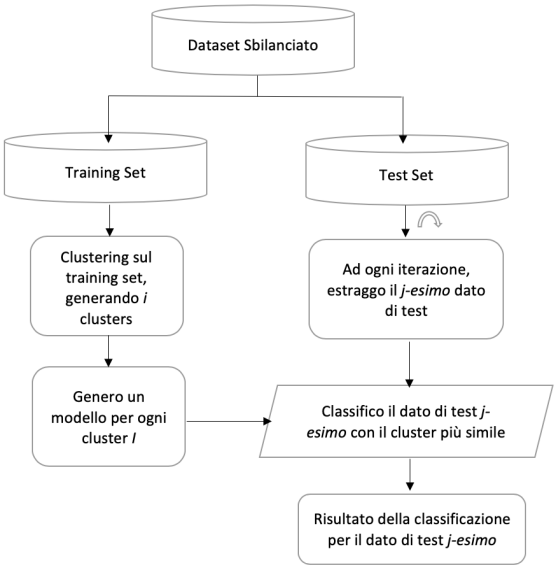


Figura 4.4: Classificazione per singolo dato di test

La parte di sinistra del flow è la medesima spiegata nella Sezione 4.3.5, ciò che varia è il procedimento raffigurato nella parte destra:

1. A seguito dell'applicazione dell'Holdout o della Cross Validation, dal dataset vengono generati il training set ed il test set;
2. Vengono generati e addestrati n modelli differenti, a seguito dell'applicazione di un algoritmo di clustering sul training set;
3. Si classifica il *j-esimo* dato di test, associando tale dato al cluster più simile.
   In questa tesi con *più simile* si intende il cluster per cui è minimizzata la distanza euclidea (vedi equazione (3.26)) tra il suo centroide ed il dato di cluster.

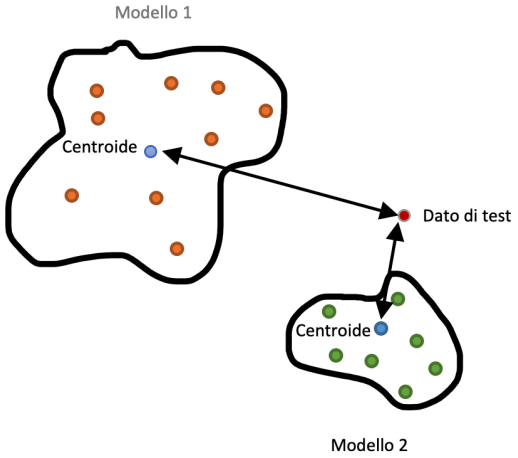


Figura 4.5: Similarità modello - data test

Nella Figura 4.5 sono rappresentati due modelli addestrati partendo dai cluster ottenuti dall'applicazione di un algoritmo di clustering.
Per classificare il dato di test j viene scelto il modello 2, in quanto il suo centroide è più vicino al dato di test.

4.3.7 Classificazione basata su cluster come dato di test

Questa metodologia presenta un approccio diverso nei confronti del test set.
Difatti il test non viene più visto come un singolo dato, ma come un cluster di dati, dove ogni cluster *j-esimo* è stato ottenuto a seguito di un algoritmo di clustering.
La Figura 4.6 dà una rappresentazione del procedimento seguito durante il processo di classificazione.

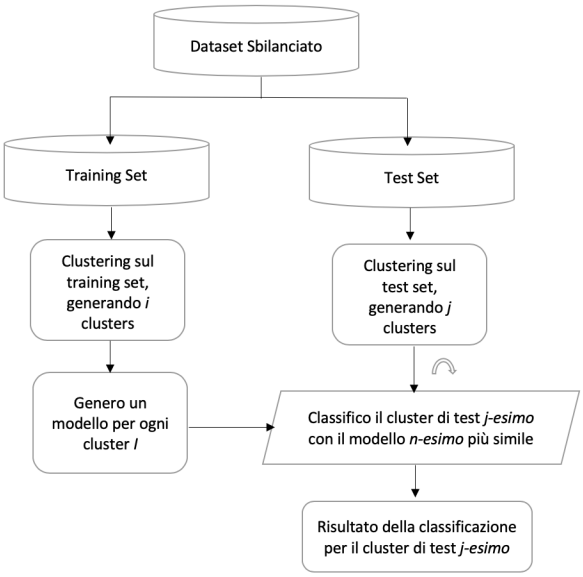


Figura 4.6: Classificazione per cluster come dato di test

1. Sul dataset iniziale è applicata la Holdout o la Cross Validation, così da ottenere il training set ed il test set;

2. Partendo dal training set, a seguito della generazione di i cluster differenti, vengono prodotti n modelli, ognuno rappresentato da un cluster;

3. Si applica l'algoritmo di clustering anche sul test set, generando così j cluster, ognuno identificato da un centroide;

4. Tutti i dati associati al cluster *j-esimo* sono classificati con il modello *n-esimo* più simile, dove con *più simile* si intende il cluster per cui è minimizzata la distanza

euclidea (vedi equazione (3.26)) tra il suo centroide ed il centroide del cluster di test:

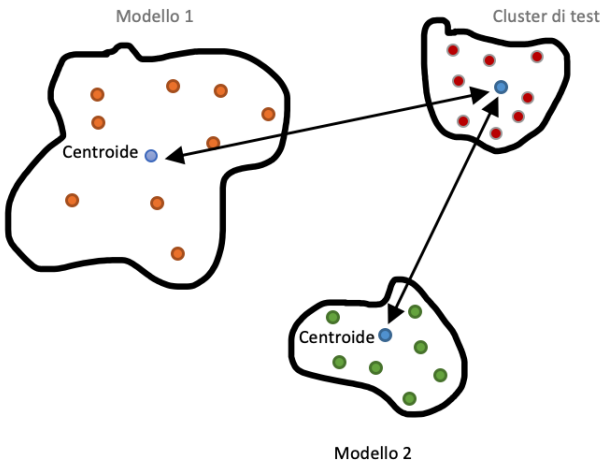


Figura 4.7: Similarità modello - cluster test

I modelli 1 e 2 della Figura 4.7, ottenuti a seguito della fase di training di ogni cluster, vengono confrontati con il cluster contenente i dati di test.
Viene scelto per la classificazione del cluster di test il modello 2 in quanto la distanza tra i rispettivi centroidi è più piccola rispetto alla distanza tra il centroide del modello 1 e il cluster di test.

4.3.8 Vantaggi e svantaggi

Gli approcci esposti nelle Sezioni 4.3.6 e 4.3.7 hanno, come già visto, il vantaggio di associare al dato di test il miglior modello possibile.
Difatti applicare il Clustering per poi effettuare la classificazione, significa utilizzare il modello più simile per caratteristiche, capace di capirne al meglio l'etichetta del dato di test.
Tuttavia, l'applicazione del Clustering pone davanti a sé un problema: la grandezza del cluster utilizzato per la fase di training.
Il numero di elementi all'interno di ogni cluster di training è un aspetto che non bisogna sottovalutare, in quanto la fase d'esecuzione potrebbe produrre dei gruppi troppo piccoli, tali da non addestrare adeguatamente un modello di classificazione. In base all'algoritmo di clustering utilizzato, la scarsità di dati in un dataset di training può essere causata da due fattori:

- Il K-Means o il DBSCAN crea dei cluster troppo piccoli.
  Ciò non si traduce sempre con un errore da parte dell'algoritmo ma, banalmente, potrebbe essere dovuto alla disposizione dei dati nello spazio;

- Il DBSCAN etichetta un gran numero di dati di training come rumore, così da ridurre la dimensione del training set.

4.3.9 Scelta dei parametri di un algoritmo di clustering

A seconda dell'algoritmo di clustering utilizzato, l'utente dovrà scegliere il valore del parametro K (nel caso del K-Means) o dei parametri $MinPts$ ed Eps (nel caso del DBSCAN).
Conoscendo la composizione del training set a nostra disposizione (ovvero presupponendo che esso non vari) sarà possibile applicare le seguenti metodologie per capire come scegliere i parametri degli algoritmi di clustering:

- Nel caso del K-Means è ottimale analizzare la distribuzione dei dati nello spazio, capendo quanti cluster sono presenti, andando così a definire un valore ottimale per K.
  Se disponiamo di un dataset con due features, questa operazione si traduce nella rappresentazione su di un piano cartesiano di tutte le entry che compongono la collezione di dati.
  Nel caso di un dataset con più di due features, l'operazione di rappresentazione sul piano cartesiano dovrà essere preceduta dall'applicazione della PCA, così da ridurre il numero di features a due.

- Nel caso del DBSCAN, scegliere i parametri $MinPts$ ed Eps sarà una scelta molto più difficile. Piccole variazioni di tali parametri possono portare a risultati differenti.

Scegliere un valore di $MinPts$ piccolo vuol dire potenzialmente creare un gran numero di cluster di piccola dimensione. Ciò non va assolutamente d'accordo con quanto detto per l'applicazione dello SMOTE Oversampling e dell'addestramento.
Scegliere $MinPts$ grande vorrà dire inglobare dentro di sé anche dei dati che in realtà costituiscono rumore e che dovrebbero essere scartati dall'algoritmo.
La scelta di Eps può esser invece dettata dall'analisi del grafico nella Figura 4.8.

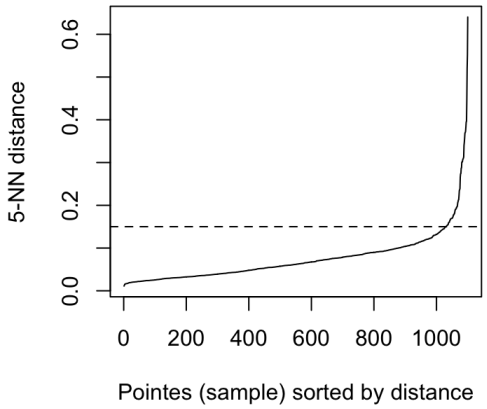


Figura 4.8: Scelta del parametro Eps

L'idea alla base nella scelta di *Eps* è che, dato un qualsiasi punto p_i, tutti i k^{th} vicini si trovino alla stessa distanza.
Al contrario, i punti di rumore staranno ad una distanza diversa dagli altri k^{th} vicini.
Supponendo di poter mettere in relazione ogni punto con la sua distanza da p_i, si ottiene un grafico come quello della Figura 4.8.
In questo caso la scelta di *Eps* ricadrà nel gradino del grafico, il quale rappresenta il valore massimo del raggio *Eps* prima che i punti vengano considerati come rumore.
Come già detto, ciò che è stato espresso finora può essere una strategia efficace nel caso che si conosca a priori come sia formato il dataset di training, così da poterne fare un'analisi dettagliata per la scelta dei valori.
Questo tuttavia non è il nostro caso, in quanto sia i dataset di training che di test cambieranno di volta in volta a seconda del risultato fornito dall'Holdout o dalla Cross Validation.
Nelle prossime prove, per rendere dinamica la valutazione di K, *Eps* e $MinPts$, verrà utilizzata la Silhouette measure.
La *Silhouette measure* [14] dà un'indicazione sulla forza di coesione tra cluster.
La misura indica, per ogni elemento, quanto esso sia lontano dagli altri cluster e quanto sia vicino al cluster di appartenenza, restituendo un interno che varia nell'intervallo [-1, +1].
Più tale misura sarà vicina a +1, più alta sarà la forza di coesione dei cluster (ovvero i cluster saranno ben formati).
Durante le prove effettuate nella sezione relativa ai risultati, il codice in Python testerà automaticamente diverse configurazioni del K-Means e del DBSCAN, scegliendo quella con Silhouette Measure più alta.

4.4 Nuove proposte per la gestione dei dataset sbilanciati

Gli algoritmi noti in letteratura forniscono un ottimo punto di partenza per l'implementazione delle proprie soluzioni.
Il nostro scopo sarà in primo luogo quello di partire dalle soluzioni già esistenti, ma allo stesso tempo verranno proposte delle metodologie alternative che hanno come base gli algoritmi allo stato dell'arte.

Potranno qui essere distinte due macro-categorie:

- *Data Level Approach*: si riprendono le soluzioni dell'Oversampling e dell'Undersampling, cercando di sfruttare il Clustering per tentare una replica dei dati d'interesse nel caso dell'Oversampling o di eliminare informazioni poco interessanti nel caso dell'Undersampling;

- *Classificazione Associativa*: verrà presentato un algoritmo di classificazione basato su Regole d'Associazione partendo dall'algoritmo associativo L^3.
  Si cercherà di migliorare la soluzione già nota associando un costo alle regole sfruttate per la predizione.

4.4.1 L'Oversampling e l'Undersampling attraverso il Clustering

In entrambe le soluzioni random a livello di dato, il grande limite è non tener conto dell'informazione correlata al dato.
È importante precisare che se nel caso dell'Oversampling tale limite può esser superato dallo SMOTE che non replica in maniera casuale, nel caso dell'Undersampling non vi sono soluzioni alternative all'eliminazione casuale di dati.
Di seguito si vorranno sfruttare gli algoritmi non supervisionati di clustering per effettuare una duplicazione (o cancellazione) mirata.

- ***Clustered Oversampling:*** gli algoritmi sopra citati non tengono in considerazione il tipo di dato su cui si sta facendo Oversampling.
  Ad esempio, potrebbe essere interessante generare delle repliche di elementi con features che sono poco frequenti all'interno del dataset così da addestrare il modello alla classificazione di tali elementi.
  Ma cosa si intende per entry con features poco frequenti?
  Nel caso del dataset spiegato durante lo SMOTE Oversampling, potremmo avere la caratteristica *speal length* ≤ 10 che ben identifica l'etichetta di classe *setosa*, ma che tuttavia è presente poche volte all'interno del dataset di training, così da essere ignorata durante la classificazione.
  Tramite un algoritmo di clustering come il K-Means o il DBSCAN, tutti i dati di training appartenenti alla classe minoritaria possono essere etichettati con un'etichetta di cluster e, i gruppi meno frequenti, replicati con ripetizione casuale degli elementi appartenenti a tale gruppo.

  Prendiamo in considerazione la Figura 4.9, dove possiamo immaginare di trovare la rappresentazione della classe minoritaria di un dataset di training:

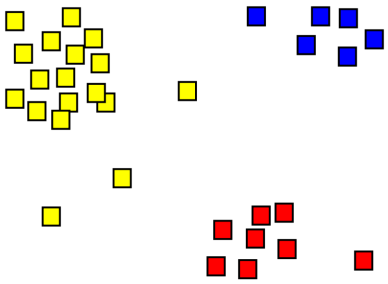


Figura 4.9: Clustered Oversampling

A seguito dell'applicazione del K-Means con $K = 3$, sono stati generati i cluster giallo, rosso e blu.
Grazie al Clustered Oversampling, verranno replicati gli elementi dei cluster blu e rosso, poiché tali caratteristiche sono meno presenti all'interno del training set.

- ***Clustered Undersampling:*** viene utilizzato un algoritmo di clustering come il K-Means o il DBSCAN per fare una scelta intelligente nella selezione dell'entry da eliminare.
  Si fa nuovamente riferimento alla Figura 4.9, immaginando ora che essa sia la rappresentazione del Clustering effettuato sulla classe maggioritaria.
  Grazie all'applicazione del Clustering, sono stati individuati tre gruppi: quadrati gialli, rossi e blu.
  Poiché i dati appartenenti alla classe gialla sono già ben rappresentati, l'algoritmo deciderà di eliminare dal training set proprio questi ultimi, così che il modello dedichi più attenzione ai gruppi rosso e blu, prevenendo così il fenomeno dell'overfitting.

A prescindere dalla soluzione adottata, l'Oversampling e l'Undersampling potrebbero avere dei limiti legati al dataset che si sta considerando per l'analisi.
Le seguenti considerazioni forniscono anche un'indicazione sul tipo di strategia da adottare in base alla dimensione e struttura del dataset.
Ad esempio, potrebbe risultare inutile l'applicazione dell'Oversampling per un dataset con un IR molto alto: questo implicherebbe la replica (o creazione) di un grande numero di dati.
Potrebbe invece essere utile l'applicazione dell'Undersampling per l'eliminazione d'informazione ridondante all'interno della classe maggioritaria.
Se l'IR è invece tendente ad uno, potrebbe essere d'aiuto l'applicazione dell'Oversampling (ragionamento opposto rispetto al precedente).
Questa è la linea seguita nel corso di questa tesi: per i dataset di Benchmark UCI con $IR \simeq 1$, sarà molto più efficiente l'applicazione dell'Oversampling.

Nel caso dell'applicazione degli algoritmi per il dataset legato al caso di studio reale con IR molto alta, verrà dimostrato che l'Oversampling sarà poco performante.

4.4.2 Lo SMOTE con valutazione e rimozione

Come visto nella Sezione 4.3.1, l'applicazione dello SMOTE permette la creazione di nuove entry a partire da quelle già esistenti nel dataset di training.
La peculiarità di questi nuovi dati sintetizzati consiste nell'essere stati generati da elementi della classe minoritaria.
Nel caso della Figura 4.1, dove i dati della classe minoritaria e maggioritaria sono ben separati, non sembrano esserci particolari problematiche nella generazione dei nuovi dati: a causa della netta distinzione tra le due classi, sarà grossomodo garantito che gli elementi che nasceranno dalla fase di sintetizzazione apparteranno alla classe minoritaria.
Tenendo conto della considerazione precedente, sarà obbligatorio rispondere alla seguente domanda: lo SMOTE garantisce che gli elementi nati in fase di sintetizzazione appartengano realmente alla classe minoritaria?
Purtroppo questo non è garantito.
La probabilità che l'elemento generato apparterà alla classe maggioritaria (invece della classe minoritaria, come da nostro obbiettivo) sarà tanto più alta, tanto più alto è il grado di coesione tra le due classi.
In fase di ricerca, sono stati svolti due tentativi al fine di eliminare gli elementi potenzialmente dannosi in fase di training.
A seguito dello SMOTE, si potrà agire tramite:

1. Rimozione degli elementi appartenenti alla classe minoritaria, generati a seguito dell'utilizzo dello SMOTE: vengono eliminati dal training set tutti gli elementi sintetizzati che potenzialmente potrebbero appartenere alla classe maggioritaria;

2. Rimozione degli elementi classificati in maniera non corretta: verranno eliminati dal training set tutti gli elementi che potenzialmente comportano una classificazione errata.

Nel primo caso, a seguito dell'Oversampling effettuato tramite lo SMOTE, vengono identificati tutti gli elementi nati dal processo di creazione.
Questi verranno sottoposti a giudizio, al fine di comprendere se apparterranno o meno alla classe minoritaria.
Per far ciò, prima di proseguire con la creazione del modello, viene creato un classificatore preliminare, con il quale verranno validati i dati sintetizzati.
Solamente quelli etichettati come appartenenti alla classe minoritaria saranno in un secondo momento utilizzati per la costruzione del modello.
Si utilizza la Figura 4.10 per spiegare dettagliatamente il processo di cancellazione.

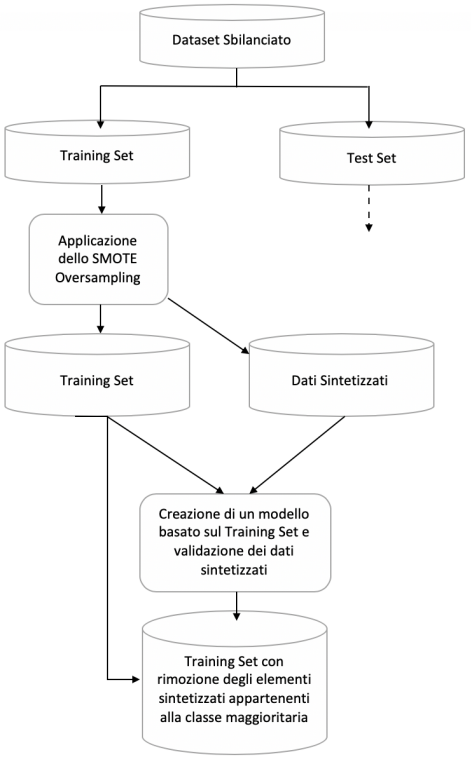


Figura 4.10: SMOTE con valutazione della classe minoritaria

1. Ai fini della validazione della nuova tecnica, il dataset di partenza viene suddiviso in training set e test set, attraverso una delle tecniche di Holdout, Cross Validation o Leave One Out Validation;

2. Viene applicato lo SMOTE Oversampling sul training set, così da generare nuovi dati sintetizzati DS_{min} appartenenti alla classe minoritaria;

3. I dati del training set di partenza TS vengono utilizzati per generare un nuovo modello, con il quale verranno successivamente validati i dati DS_{min};

4. I soli dati DS_{min} considerati dal classificatore come appartenenti alla classe minoritaria saranno accoppiati al training set di partenza TS, per formare quello che sarà il modello vero e proprio utilizzato per la classificazione finale.

Come verrà dimostrato nella sezione inerente ai risultati sperimentali, questa metodologia consentirà di migliorare le prestazioni dello SMOTE.

A seguito di questa prima analisi, può esser proposta un'altra considerazione: è garantito che i dati utilizzati durante la fase di training appartengano alla classe con la quale sono stati etichettati, poiché essi provengono dalla collezione di dati originale.
Ma tali dati come verranno considerati all'interno del modello?

Verranno considerati con l'etichetta di classe corretta o produrranno una misclassificazione?
Ispirati dal funzionamento dell'algoritmo associativo L^3, la nuova proposta consiste nell'eliminare dal training set tutte quelle entry che potenzialmente genereranno falsi positivi o falsi negativi.
In sostanza questo può esser fatto costruendo e validando un modello sullo stesso set di dati.
Si noti che questa considerazione non sarà valida nel solo caso dello SMOTE, ma in un qualsiasi contesto di predizione.
Tuttavia, in questa tesi, nella quale la semplice applicazione di un algoritmo di classificazione diventa vana, questa metodologia avrà senso solamente a seguito dell'applicazione dello SMOTE.
Si fa riferimento alla Figura 4.11 per spiegare nel dettaglio l'utilizzo di questa tecnica.

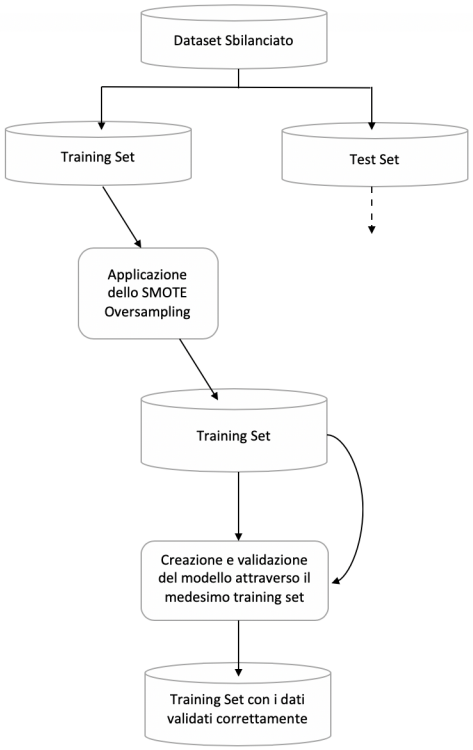


Figura 4.11: SMOTE con valutazione del dataset

1. Il dataset d'origine viene suddiviso in training set e test set, secondo una delle tecniche di splitting esposte nei paragrafi precedenti;

2. Viene applicato lo SMOTE Oversampling sul training set d'origine, generando un nuovo training set TS_{SMOTE};

3. Viene costruito un modello di classificazione sul training set TS_{SMOTE}.
   A seguito della fase di training, il modello viene validato con $Test\ Set = TS_{SMOTE}$.
   I dati validati correttamente, verranno mantenuti per formare un nuovo training set, denominato TS_{rem}.

4. TS_{rem} verrà utilizzato per la creazione del modello d'interesse.

Questa tecnica, almeno in fase teorica, sembrerebbe comportare un sostanziale miglioramento in quanto fa sì che vengano esclusi dal modello quei dati che partecipano ad una classificazione errata, sulla stessa linea di L^3.
Tuttavia, come si mostrerà nella sezione sperimentale, tale tecnica sembra essere molto meno efficiente della metodologia che effettua una valutazione della sola classe minoritaria.
La ragione può esser intravista nel concetto di creazione del modello: non si agirà sul classificatore finale, ma solamente sui dati che lo andranno a comporre.
Anche se saranno eliminati nello step iniziale di classificazione dei dati che già in prima battuta vengono classificati in modo non corretto, non è garantito che con la formazione del secondo modello non si verifichi nuovamente la problematica.
Si noti che L^3 agisce sul modello stesso, in questo caso si agisce solamente sui dati utilizzati per la fase di training.

4.4.3 Classificazione tramite Regole Associative

È già stato spiegato nella Sezione 3.1.4 cosa si intende per Regole d'Associazione.
In questo capitolo si vuol sfruttare la conoscenza estratta dalle Regole Associative per creare un *classificatore associativo cost-sensitive* ad hoc per dataset sbilanciati. Questo, al contrario di quanto già fatto finora, è stato implementato in linguaggio Java.
Tale scelta è dettata da due motivazioni:

1. Il package Scikit-Learn non contiene nessun algoritmo (e.g. APRIORI) capace di effettuare l'estrazione delle Regole d'Associazione da un dataset.
   Al contrario Java, con la sua libreria Weka [15], offre una vasta gamma di algoritmi di Machine Learning utili per l'implementazione del classificatore associativo.

2. Viene sfruttato il classificatore basato su regole L^3 per estrarre solamente le regole di maggiore rilevanza.

Il funzionamento del classificatore associativo può essere suddiviso in tre macro-aree:

1. Estrazione delle regole dal dataset di training grazie all'algoritmo L^3;

2. Associazione di un costo ad ogni regola estratta (fase di training);

3. Classificazione del dataset di test.

4.4.3.1 L'estrazione delle regole: l'algoritmo L^3

In fase di sviluppo, ai fini dell'estrazione delle Regole Associative, è stato scelto l'utilizzo dell'algoritmo *Live and Let Live* (L^3) [16].
La scelta di L^3 si giustifica con le seguenti motivazioni:

- Le regole vengono salvate in maniera *compatta*.
  Tale metodologia permette di riscriverla, insieme alla sua etichetta e alle proprie misure di qualità (confidenza, supporto, lift).
  Si può passare in qualsiasi momento dalla forma compatta alla vera regola.
  L'algoritmo di classificazione sviluppato utilizza per la classificazione le regole in forma compatta.
  In un successivo momento (in fase di visualizzazione) si passa dalla forma compatta alla forma di rappresentazione normale;

- L'algoritmo permette l'estrazione e l'utilizzo delle regole a maggior rilevanza.
  Altri classificatori associativi come CAEP [17], CMAR [18], CBA [19], e ADT [20], utilizzano per la classificazione tutto il set di regole estratte.
  Come succede già con APRIORI, l'unico metodo disponibile per migliorare la qualità della regola è la scelta di una soglia di confidenza e supporto alta.
  È importante notare che tale scelta può cambiare la qualità delle regole estratte, ma la misura non è assolutamente correlata con la qualità della classificazione.
  L^3 utilizza in parte una soglia di supporto, confidenza e lift per l'estrazione delle regole, ma genera due subset di regole *"buone"*, servendosi di un approccio denominato *Lazy Pruning*.

Con *Lazy Pruning* si intende una fase di pruning che segue quella d'estrazione delle regole.
Come già detto, un primo stadio si ha tramite la scelta di una certa soglia di supporto, confidenza e lift.
In fase di sviluppo sono state fatte le seguenti scelte d'implementazione:

- Supporto $s = 5.0$;
- Confidenza $C = 50.0$;
- Scelta delle regole positivamente correlate.

In una seconda fase di pruning, le regole sono divise in tre liste differenti. La divisione avviene utilizzando quelle del training set $t \in T$ impiegate per la costruzione del modello:

- ***Regole Utilizzate:*** lista di regole che classificano correttamente almeno una regola t del training set T;
- ***Regole di Scorta:*** gruppo di regole che non classificano nessun dato del training set.
  Queste, tuttavia, potrebbero essere adoperate nella fase di classificazione del test set;

- ***Regole Dannose:*** lista contenente tutte le regole che classificano solamente in modo non corretto le regole t del training set T.
  Data la mancata classificazione del training set, può essere lecito pensare che tali regole commetteranno errori anche nella classificazione del test set, perciò sarà opportuno scartarle.

Definiamo la lista delle *Regole Utilizzate* come ***Regole di Livello 1*** (R_{l1}).
Le *Regole di Scorta* vengono definite come ***Regole di Livello 2*** (R_{l2}).
L'algoritmo L^3 *Cost Sensitive* tenterà in prima fase l'utilizzo della lista R_{l1}.
Nel caso non fossero presenti regole utilizzabili, ci si servirà della lista R_{l2}, come spiegato nella Sezione 4.4.5.

4.4.4 La classificazione Cost-Sensitive

Come anticipato nell'introduzione, L^3 *Cost Sensitive* può essere definito come un classificatore Cost-Sensitive.
Con *Cost Sensitive Classification* intendiamo una tipologia di classificazione che tiene conto del costo associato a una classificazione errata.
Questa strategia può essere utile nell'ipotesi in cui si voglia dare più importanza alla classificazione di una classe rara, proprio come nel caso di quella minoritaria nel contesto di un dataset sbilanciato.
Per far ciò, verrà assegnata ad ogni regola un costo, seguendo la matrice riportata nella Tabella 4.1:

Tabella 4.1: Matrice di Costo

Reale/Predetta	**Classe = +**	**Classe = −**
Classe = +	−1	100
Classe = −	1	0

La matrice rappresentata nella Tabella 4.1 è detta *Matrice di Costo*, ed è popolata considerando la *Classe* $= +$ come rara, e la *Classe* $= -$ come classe di maggioranza.

Definiamo come $C(i, j)$ il costo associato alla classificazione di un elemento di classe i come appartenente alla classe j. Ad esempio, $C(+, -)$ denota il costo di una previsione errata, dove un'entry appartenente alla *Classe* $= +$ è stata prevista come appartenente alla *Classe* $= -$.
Lo scopo in questo caso non sarà associare un costo alla singola classificazione (che può essere errata o meno), ma ad una regola.
Il costo associato alla regola R può essere valutato come segue:

$$C(R) = TP \times C(+,+) + FP \times C(-,+) + FN \times C(+,-) + TN \times C(-,-) \quad (4.2)$$

I valori TP, TN, FP e FN rappresentano il numero di entry predette in modo corretto o errato da una certa regola.

Capiamo così che il costo di R sarà associato alla qualità della classificazione, con particolare riferimento alla classe minoritaria: se la regola è ben capace di classificare dei dati appartenenti alla classe minoritaria, essa avrà costo basso. Viceversa, nel caso di classificazione errata, il costo della regola sarà alto.
Chiaramente in fase di classificazione verranno scelte le regole a costo basso.
Normalmente in un dataset sbilanciato il costo della classe positiva sarebbe molto più alto del costo di quella negativa: la Matrice di Costo inverte la rotta, riducendo il costo associato alla classe rara.

4.4.5 Assegnazione del costo e classificazione

Di nostro interesse sarà assegnare ad ogni regola estratta mediante l'algoritmo L^3 un costo, valutandolo attraverso l'utilizzo della formula (4.2).
Come spiegato nella Sezione 3.2.3, una regola è un'espressione nella forma:

$$R : X \implies Y$$

Nel caso particolare della classificazione, la testa della regola Y rappresenta l'attributo da classificare.
Definiamo ora le variabili:

- R: set di regole estratte;
- T: set di training;
- *cost*: costo associato ad una regola;
- *pc*: denota la classe positiva.

L'algoritmo d'assegnazione del costo ad un set di regole può essere rappresentato dal seguente pseudocodice:

1. **for each** $r \in R$ **do**
2. $tp, fp, fn, tn = 0$
3. **for each** $t \in T$ **do**
4. $tp+ = |\{t|r.X \subseteq t.X \wedge r.Y \in t.Y \wedge t.Y \in pc\}|$
5. $tn+ = |\{t|r.X \subseteq t.X \wedge r.Y \in t.Y \wedge t.Y \notin pc\}|$
6. $fp+ = |\{t|r.X \subseteq t.X \wedge r.Y \notin t.Y \wedge t.Y \notin pc\}|$
7. $fn+ = |\{t|r.X \subseteq t.X \wedge r.Y \notin t.Y \wedge t.Y \in pc\}|$
8. **end for**
9. $r.cost = tp \times cost.C(+,+) + fp \times cost.C(-,+)$
   $+fn \times cost.C(+,-) + tn \times cost.C(-,-)$

10. **end for**

Analizzando l'algoritmo è possibile notare come ad ogni regola estratta dal training set venga assegnato un valore relativo al costo di classificazione corretta (TP e TN) ed errata (FP e FN).
Se la regola è capace di classificare i dati del training set (molti TP o TN, con $C(+,+)$ e $C(-,-)$ negativo o nullo), ad essa sarà associato un costo basso, viceversa se la regola classifica in modo errato (molti FP o FN con $C(+,-)$ e con $C(-,+)$ positivi, specialmente nel caso del costo relativo all'errore sulla classe maggioritaria), avrà un costo alto. Tale indicazione torna molto utile durante la fase di classificazione.

Dopo aver applicato la funzione di costo a tutte le regole, il modello di classificazione è pronto per l'utilizzo dei dati del test set non ancora etichettati.
Definiamo le variabili:

- R_{l1}: set di regole estratte di livello 1, a cui è stato precedentemente assegnato un costo;
- R_{l2}: set di regole estratte di livello 2, a cui è stato precedentemente assegnato un costo;
- T: test set, contenente l'etichetta di classe Y non ancora valorizzata;
- $default\_class$: classe di default, utile se nessuna regola riesce a classificare il dato di test $t \in T$;
- c_p: classe positiva;
- c_n: classe negativa.

Il funzionamento di classificazione può essere riassunto nel seguente pseudocodice:

1. $default\_class = c_p$
2. **for each** $t \in T$ **do**
3. $t.Y = default\_class$
4. Create list $R_{ml1} = \{r | r.X \subseteq t \wedge r \in R_{l1}\}$
5. Create list $R_{ml2} = \{r | r.X \subseteq t \wedge r \in R_{l2}\}$
6. Sort R_{ml1} by cost
7. Sort R_{ml2} by cost
8. **if** (size(R_{ml1})!=0) **begin**
9. $class\_lower\_risk$ = get element of R_{ml1} in position 0

10. $t.Y = class\_lower\_risk$

11. **end if**

12. **else if** (size(R_{ml2})!=0) **begin**

13. $class\_lower\_risk$ = get element of R_{ml2} in position 0

14. $t.Y = class\_lower\_risk$

15. **end if**

16. **end for**

La funzione di classificazione viene applicata per ogni entry di test $t \in T$.
In fase di sviluppo, è stato notato che non si può dare per scontato che esista sempre una regola capace di classificare un dato di test t.
Questo sarà plausibile per due motivi:

1. In fase d'estrazione delle regole è stata scelta una soglia di confidenza o supporto troppo alta, così che la regola capace di classificare il dato t sia stata eliminata;

2. Il dato di test t appartiene alla classe rara, per tal motivo nel training set non esistono abbastanza entry capaci di generare una regola che riesca a classificare t.

Si vuol evidenziare la possibilità per cui un dato di test risulti non classificato, per mancanza di una regola che lo possa etichettare opportunamente: poiché durante la fase d'esecuzione l'ipotesi 2 sarà molto probabile, è stato scelto di utilizzare come classe di default la classe rara (nel nostro caso c_p), cosicchè essa possa essere assegnata in caso di mancato matching con una regola di training.
Si noti che si sarebbe potuto lasciare il dato come non etichettato: tuttavia non è stata intrapresa questa scelta in quanto si vuol implementare un classificatore, dove non è una buona opzione tenere un dato di test nuovamente incognito.
Terminata questa prima fase di configurazione iniziale, verranno estratte tutte le regole capaci di classificare il dato t, ovvero per cui sarà soddisfatta la condizione $r.X \in t$ (il corpo della regola sarà contenuto nel dato di test t).
Tali regole verranno inserite all'interno delle liste R_{ml1} e R_{ml2}, a seconda che r appartenga alla lista R_{l1} o R_{l2}.
Entrambe le liste verranno successivamente ordinate per costo $r.cost$.
Si ricorda, come detto precedentemente, che il nostro scopo sarà utilizzare la regola a costo più basso.
A questo punto si è pronti per la classificazione: se sarà presente almeno una regola all'interno della lista R_{ml1}, quella in testa alla lista ordinata sarà usata per la classificazione, altrimenti si sfrutterà la lista R_{ml2}, prendendo anche in questo caso la regola in testa. Se anche la lista R_{ml2} sarà vuota, il dato incognito verrà etichettato con la classe di default.

4.5 Il software per la classificazione

Nel corso di questa tesi, sono state e saranno affrontate diverse problematiche legate alla classificazione di dataset sbilanciati, per le quali sono state esposte diverse soluzioni.
Si è resa di fondamentale importanza la necessità di fornire alla compagnia assicurativa con la quale è stata svolta parte di questa tesi un software di classificazione facilmente utilizzabile dall'esperto di dominio, che potenzialmente ha una conoscenza limitata sull'utilizzo di Python.
A seguito della fase di sviluppo, si disporrà di un software Python configurabile ad hoc con il quale sarà possibile gestire tutte le operazioni di Machine Learning analizzate finora.
Al fine di rendere utilizzabile il prodotto software finale, è stata sviluppata un'interfaccia grafica (*GUI*) che faciliti l'esecuzione di tutte le fasi del processo di Data Mining.
Per il suo sviluppo si è scelto l'utilizzo di Java e del suo plug-in *Window Builder* [21].
Window Builder è un plug-in dell'IDE Eclipse che permette di creare facilmente un'interfaccia grafica, mettendo a disposizione una GUI che consente lo spostamento manuale di oggetti visivi dal proprio pannello applicativo.
Rispetto a qualsiasi altro tipo di linguaggio per lo sviluppo grafico, Java offre la possibilità di rendere l'applicazione *cross platform*, ovvero utilizzabile in qualsiasi ambiente. L'impiego di altri linguaggi, come ad esempio *C#*, avrebbe ridotto l'utilizzo del software al solo ambiente Windows.
Per lo sviluppo grafico è stato utilizzato il framework *Swing* di Java.
Swing è la naturale evoluzione di *Java AWT*, che permette di creare una finestra grafica (oggetto della classe *JPanel*) su cui è possibile inserire diversi componenti, come etichette (*JLabel*), caselle di input (*JTextEdit*), bottoni (*JButton*) e tanto altro ancora.
Durante la fase di progettazione, si è fatto riferimento alla documentazione ufficiale di *Swing* [22], contenente tutte le informazioni necessarie sui componenti grafici utilizzati.
L'applicativo su cui viene eseguito il software, deve presentare i seguenti requisiti minimi:

- Sistema Operativo: Windows 7/8/10, Linux o MacOS;
- Librerie installate: Oracle Java SE 8 o superiore;
- Python 2.7 o superiore.

Si noti che mentre il front-end dell'applicativo è costruito in Java, il back-end è sviluppato in Python, così da permette l'utilizzo degli algoritmi di Machine Learning citati in questa tesi.
Com'è possibile un utilizzo combinato di Java e Python che sono linguaggi totalmente diversi?
Si è optato per l'uso della classe *Runtime*.
Questa classe crea un oggetto che contiene un riferimento all'ambiente in cui l'applicazione è in esecuzione.
Questo uso permette, tramite il comando *exec()*, l'esecuzione di funzionalità di terze parti.

In altri termini, gli script Python vengono visti come programmi esterni, eseguibili tramite il comando *Python*.
Si riporta un breve estratto di codice:

1. Runtime run = Runtime.getRuntime();
2. String target = new String("python ./pyscript/MainApp.py");
3. run.exec(target);

Il codice di cui sopra, permette l'esecuzione dello script MainApp.py python.
Nel nostro caso, tale script (opportunamente parametrizzato) consentirà l'esecuzione dei diversi algoritmi di Data Mining.

Nel seguito verranno analizzate tre funzionalità principali, per ognuna delle quali si dedicherà una sezione apposita: la fase di preprocessing, l'estrazione delle regole d'associazione ed infine la classificazione.

4.5.1 Il software di preprocessing

La parte grafica del software di preprocessing (come per la parte di back-end) è stata oggetto di un altro lavoro di tesi.
In questa tesi si intende dare una breve introduzione su quali sono le funzionalità offerte, ma non si vuole assolutamente entrare nel merito della fase di sviluppo.
Al fine di spiegare l'utilizzo dell'applicativo, si fa riferimento alla Figura 4.12.

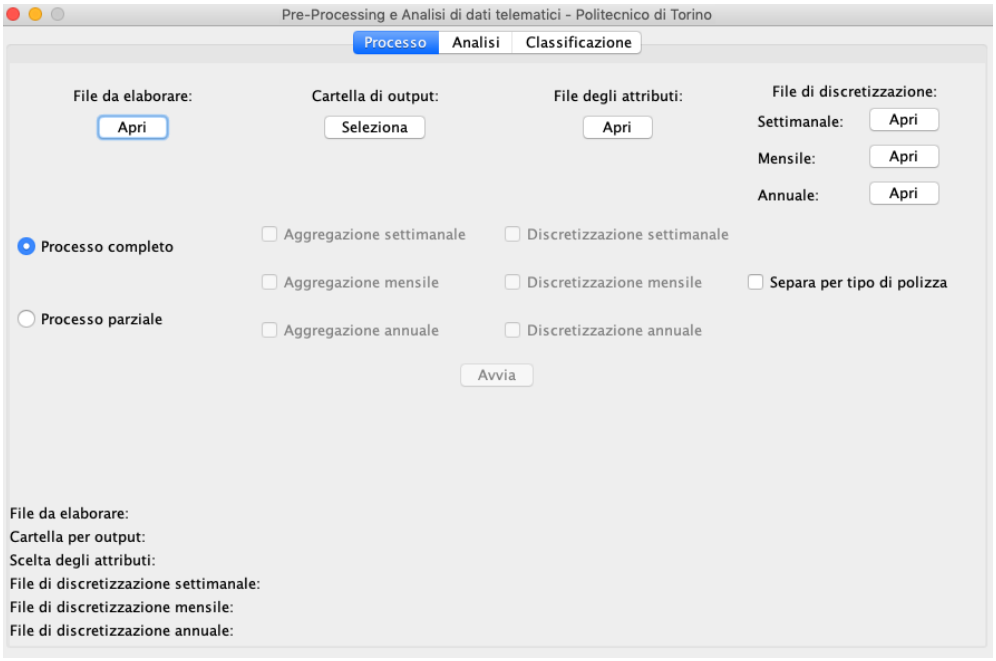


Figura 4.12: Tab di preprocessing

Dall'osservazione di tale figura, possono essere identificati quattro fattori principali per cui, prima di effettuare qualsiasi altro tipo di analisi, è necessario l'utilizzo di questa sezione:

- *Scelta della granularità*: come già spiegato nella Sezione 5.2, si permette il passaggio da una granularità giornaliera ad una granularità mensile e settimanale.
- *Gestione dei dati mancanti:* alcune righe del dataset di partenza presentano dei valori mancanti.
  Tali valori possono essere semplicemente collegati ad una mancata rilevazione da parte della scatola telematica.
  In questa parte, i dati mancanti vengono eliminati (rappresentano una minoranza trascurabile) così da evitare errori durante la parte d'analisi per la quale non si troverebbero dei dati d'interesse.
- *Discretizzazione del dataset:* a partire dai valori continui viene generato un dataset formato da range discreti.
- *Selezione degli attributi d'interesse:* vengono estratti dal dataset di partenza solamente gli attributi d'interesse.
  Tali attributi possono essere scelti attraverso un apposito file di configurazione.

Il bottone *file da elaborare* permette di selezionare il file preso in carico per il preprocessing.
Il risultato dell'elaborazione verrà salvato nella cartella di output specificata tramite il bottone *Cartella di output*.
La funzionalità di scelta *File degli attributi* permette di specificare un elenco di attributi che prenderanno parte al dataset finale.
Infine, può essere effettuata la scelta del *file di discretizzazione* grazie ai bottoni presenti al di sotto dell'etichetta *File di discretizzazione*.
In tale file di configurazione verrà specificato il valore discreto da assegnare a seconda del valore continuo assunto dall'attributo.
Sarà possibile selezionare file diversi a seconda dell'aggregazione scelta.

Viene lasciata la possibilità all'utente di decidere se avere tutte le aggregazioni possibili, selezionando l'opzione *Processo Completo*, o scegliere quali aggregazioni ottenere, selezionando l'opzione *Processo Parziale* e le operazioni d'interesse da applicare.

4.5.2 L'estrazione delle regole d'associazione

Per un esperto di dominio è necessario riuscire ad estrarre conoscenza dalla propria base dati.
Una soluzione possibile si può trovare nell'estrazione delle regole d'associazione.
Di particolare interesse sono le regole il cui conseguente corrisponde ad un indice di rischio, poiché in tal modo si capisce quale può essere la relazione tra un attributo dell'assicurato e il rischio a cui esso è esposto.

In questa tesi, lo scopo non sarà comprendere il significato che può essere estrapolato dalle regole d'associazione, ma ci si vuol limitare allo sviluppo dell'algoritmo d'estrazione e visualizzazione delle stesse.
Sfruttando l'algoritmo L^3 (vedi Sezione 4.4.3.1) è possibile estrarre le regole d'associazione che presentano come conseguente un indice di rischio e soddisfano le soglie di supporto, confidenza e lift specificate.
Il procedimento utilizzato per l'estrazione è il medesimo di quanto spiegato in precedenza, con due differenze:

- L'algoritmo viene fermato alla parte d'estrazione delle regole.
  Queste ultime, piuttosto che esser utilizzate per la classificazione, vengono esposte a video;

- Sono prese in considerazione sia le regole di livello 1 che le regole di livello 2.
  Si ricorda che in questo momento non ci interessa quanto una regola sia buona per la classificazione.

Per spiegare il processo d'estrazione, si fa riferimento alla Figura 4.13:

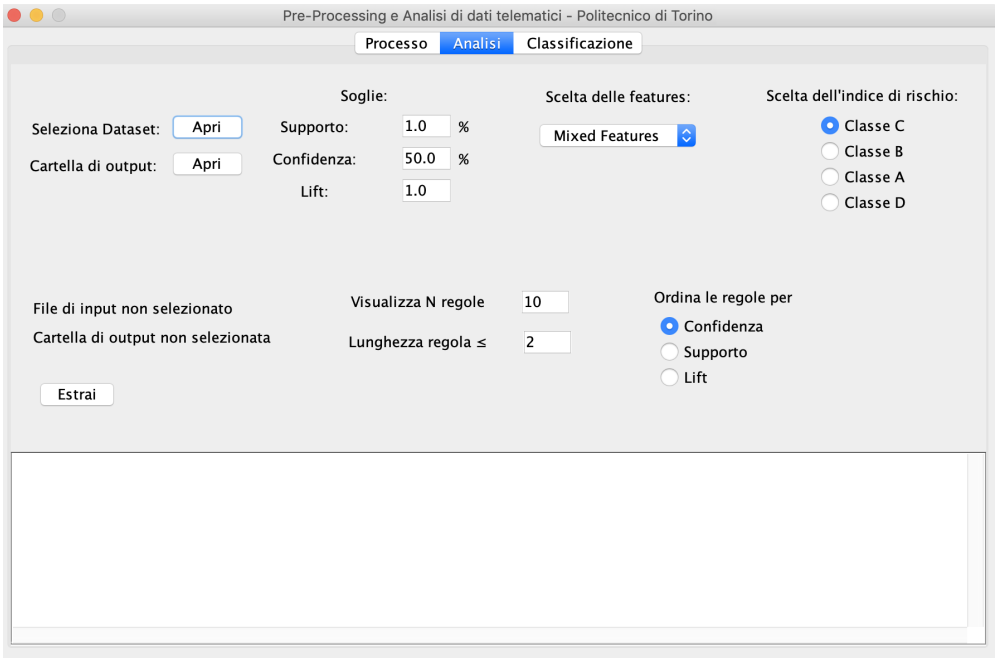


Figura 4.13: Tab d'estrazione delle regole

Come primo passo sarà necessario selezionare il file da dove verranno estratte le regole.
Ciò si può ottenere premendo il bottone *Apri* relativo alla voce *Seleziona Dataset*. Allo stesso modo, sarà possibile selezionare una cartella di output (voce *Cartella di output*) nella quale sarà salvato un file di tipo .txt contenete tutte le regole estratte.
Nella sezione *Soglie*, sarà possibile scegliere il livello di confidenza, supporto e lift che verrà utilizzato durante il processo d'estrazione delle regole.

Tutte le regole con valori di confidenza, supporto e lift minori delle soglie scelte, non verranno estratte. Per limitare il numero di features che concorrono durante la loro estrazione, sarà possibile selezionare un gruppo di attributi che faranno parte del corpo della regola. Tale gruppo di features potrà essere selezionato dal menù a tendina *Scelta delle features*.
Potranno essere scelti gli attributi inerenti a:

- *Percorrenze:* attributi inerenti al percorso effettuato dall'utente a bordo della sua vettura;
- *Stili di guida:* riporta delle caratteristiche inerenti allo stile di guida che mantiene l'assicurato;
- *Caratteristiche polizza:* attributi inerenti al tipo di polizza stipulata tra il cliente e la compagnia assicurativa;
- *Mixed Features:* una selezione mista di attributi inerenti a percorrenze, stili di guida e caratteristiche delle polizze;
- *Scegli da file:* l'utente potrà specificare, tramite un file di configurazione, tutti gli attributi utilizzati nella fase d'estrazione delle regole e che potenzialmente faranno parte del corpo della regola. Il file dovrà contenere, per ogni riga, il nome dell'attributo da selezionare.

Infine, dalla sezione *Scelta dell'indice di rischio*, sarà possibile scegliere l'indice di rischio che comporrà la testa della regola.
Date le quattro etichette di classe d'interesse, sarà possibile effettuare una scelta tra gli indici Classe A, Classe B, Classe C e Classe D.
Si rimanda alla Sezione 5.2 per una spiegazione dettagliata sul significato degli indici elencati precedentemente.
Dopo aver completato la fase di configurazione, tramite il bottone *Estrai* sarà possibile avviare l'estrazione delle regole.
Tale processo potrebbe richiedere del tempo (anche 5 minuti).
Tale tempo dipende dalla dimensione del dataset, dal numero di attributi coinvolti nell'estrazione e dalle soglie di supporto, confidenza e lift utilizzate. Più alte saranno le soglie, minore sarà il tempo d'esecuzione richiesto.
Una volta terminata l'estrazione, tutte le regole saranno salvate nella cartella selezionata dall'utente.

Tramite il pannello presente all'interno del software, sarà possibile visualizzare in modo efficace un subset delle regole estratte.
L'utente potrà selezionare:

- Il numero di regole da visualizzare a video;
- La lunghezza massima delle regole visualizzate;

- L'ordine di visualizzazione.
  Sarà possibile una scelta d'ordinamento tra le seguenti:
  - ◇ Supporto
  - ◇ Confidenza
  - ◇ Lift

Attenzione: le ultime funzionalità elencate cambiano, in real-time, il modo in cui vengono visualizzate le regole.
Tali configurazioni non incidono in alcun modo sulla modalità d'estrazione e salvataggio delle regole.

4.5.3 La classificazione

Il pannello *Classificazione* del software offre la possibilità di effettuare, su un dataset discretizzato annualmente, la Cross Validation e la Leave One Out Validation.
Viene offerta anche la possibilità di etichettare, con uno dei quattro livelli di rischio (Classe A, Classe B, Classe C e Classe D) concordati con l'esperto di dominio, un dataset non ancora etichettato.
Tale opzione è fondamentale nel caso in cui si volesse inquadrare l'utente in base alle proprie generalità ed al proprio stile di guida, al fine di proporgli un certo costo per la polizza che andrà a stipulare con l'azienda.
Come si può notare, questo pannello conclude il processo di Data Mining della Figura 2.1: partendo dal dataset proveniente dalle rilevazioni della compagnia assicurativa si riesce ad estrarre la conoscenza desiderata. Al fine di spiegare il corretto funzionamento del software di classificazione, si farà riferimento alla Figura 4.14.

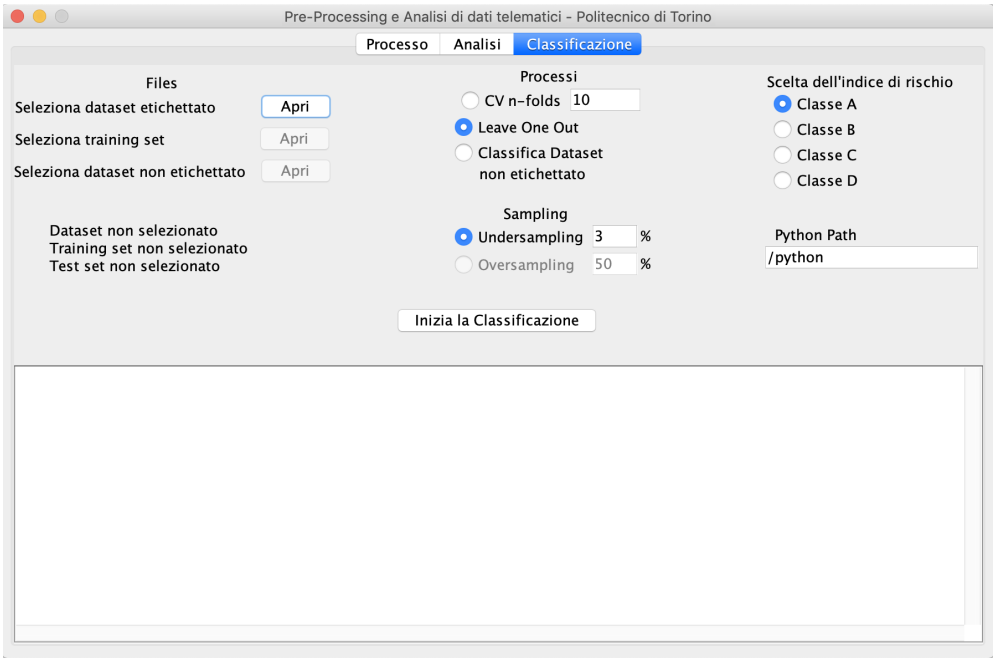


Figura 4.14: Tab di Classificazione

La sezione *Files* permette di selezionare i dataset di input.
Possono essere selezionati tre tipi di dataset, a seconda del tipo di classificazione che l'utente vorrà eseguire:

- *Dataset:* collezione di dati già etichettata con gli indici di rischio.
  Tale file dovrà essere selezionato nel caso in cui si voglia effettuare la Cross Validation o la Leave One Out Validation per testare le performance del classificatore.
- *Training Set:* collezione di dati contenente le informazioni sugli assicurati e gli indici di rischio.
  Tale file dovrà essere selezionato nel caso in cui si voglia effettuare la classificazione di un dataset non ancora etichettato (sarà il training set del classificatore).
- *Dataset non etichettato:* collezione di dati sprovvista di etichetta di classe.
  Il classificatore assegnerà un'etichetta di classe ad ogni polizza presente in questo dataset.

La sezione *"Processi e Sampling"* permette di selezionare il tipo di classificazione che si vorrà effettuare e la strategia di sampling (Oversampling o Undersampling con relative percentuali) che verranno attuate al fine di migliorare le prestazioni del classificatore.
In base al tipo di classificazione scelto, il software abiliterà o meno le strategie di sampling.
Questo comportamento è dato dalla poca utilità di applicare la Leave One Out Validation quando viene effettuato l'Oversampling sul dataset di training e viceversa, nel caso della Cross Validation con Undersampling.

L'opzione *"Classifica Dataset"* darà all'utente la possibilità di classificare un dataset sprovvisto di etichetta di classe.
In tal caso possono essere abilitate entrambe le strategie di sampling.
Il consiglio è di abilitare la strategia dell'Undersampling, in quanto più performante.

Nella sezione *"Scelta dell'indice di rischio"* viene scelta l'etichetta di classe che verrà assegnata all'entry non etichetta del test set.
Ad ogni iterazione della classificazione sarà possibile scegliere una sola etichetta di classe.
Nel caso in cui si fosse interessati a conoscere più di un indice di rischio, il processo di classificazione dovrà essere eseguito più volte. Come per la fase d'estrazione delle regole, sarà possibile scegliere un indice di rischio tra Classe A, Classe B, Classe C e Classe D.

Per permettere una corretta esecuzione degli script Python sarà necessario inserire, all'interno della text box *Python Path*, il path d'installazione di Python.
Questo può variare in base alla macchina e al Sistema Operativo in cui è in esecuzione il software, per questo motivo il setting di tale parametro è lasciato aperto all'utente.
In ogni caso, il software di classificazione proverà a capire autonomamente il path d'installazione.

Si invita l'utente al cambiamento di tale parametro solamente in caso di mancato funzionamento.

Il bottone *Inizia la classificazione* dà la possibilità all'utente di avviare il processo di classificazione.
Questo, in base alla tipologia di validazione scelta ed alla grandezza del dataset, potrebbe essere un processo con alto tempo d'esecuzione.
Per processi onerosi, una barra di progresso (Figura 4.15) darà un'indicazione del tempo rimanente.

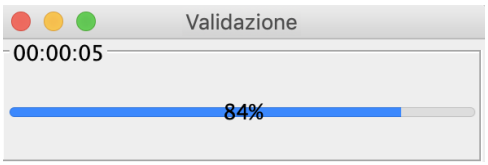


Figura 4.15: Barra di progresso

Sarà anche possibile, attraverso il bottone *Interrompi la classificazione* (presente solamente quando un processo di classificazione è in corso) interrompere l'esecuzione.
Poiché l'Undersampling e l'Oversampling hanno lo scopo di portare, in fase di training, il numero di elementi della classe *Incidente SI* e *Incidente NO* ad un egual numero, si consiglia sempre l'utilizzo dell'Undersampling con una percentuale pari al 5% (come si dimostrerà nella fase di commento dei risultati sperimentali nel Capitolo 6).
Utilizzare valori diversi da quelli indicati significherebbe perdere il beneficio dato dalle strategie di sampling implementate.
L'utilizzo dell'Oversampling è stato inserito per completezza.

Nel caso in cui fosse stata scelta una strategia di validazione tra Cross Validation e Leave One Out Validation, la console presenterà a video un output come quello proposto nella Figura 4.16.

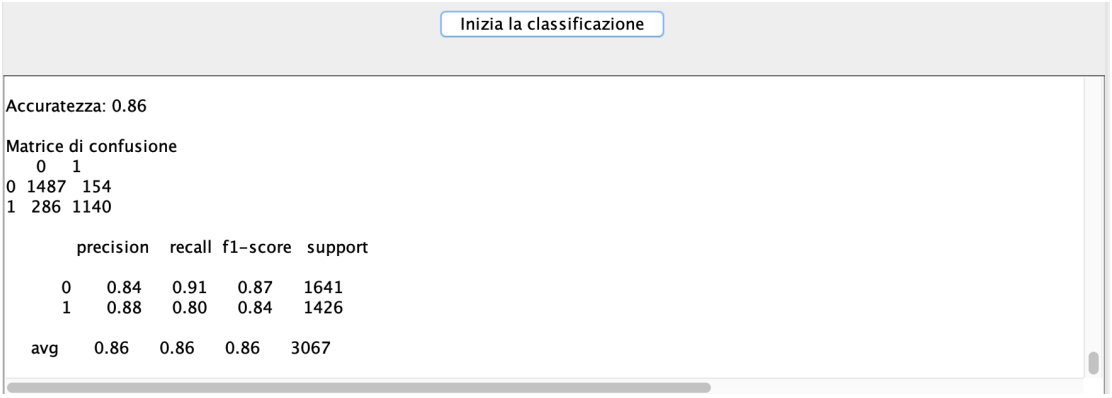


Figura 4.16: Console Python per CV e LOO

Verranno mostrate la matrice di confusione, l'accuratezza, la precisione, il richiamo

e l'F1-Score per il dataset classificato.

Nel caso di classificazione di un dataset non etichettato (opzione *Classifica dataset non etichettato*), la console fornirà delle statistiche riguardanti la classificazione (e.g. totale di entry classificate, totale classificato come *Incidente SI*, totale classificato come *Incidente NO*) e il path in cui verrà salvato il nuovo dataset contenente la relazione Numero Polizza - Etichetta di classe.
Un esempio di output è esposto nella Figura 4.17:

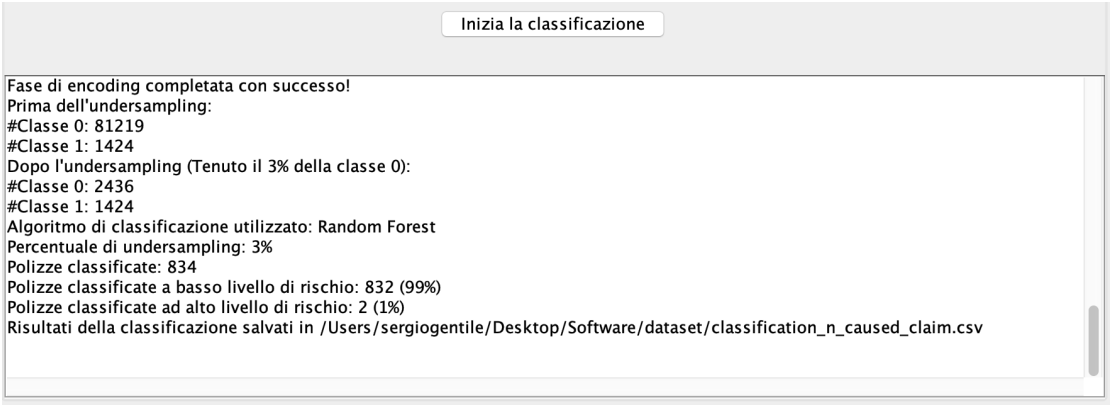


Figura 4.17: Console Python per classificazione di un dataset

5 Dataset sbilanciati in esame

Questa tesi nasce con lo scopo di classificare una collezioni di dati reali, fornita da una compagnia assicuratrice.
La caratteristica di tale dataset, in relazione con il caso di studio a cui esso è correlato, è l'essere etichettato come un dataset sbilanciato (IR molto maggiore di 1).
Data la difficoltà nella gestione di tale problematica, è stata portata avanti una fase di ricerca al fine di individuare possibili soluzioni alla gestione del problema.
A tal proposito, si è scelta la validazione delle proprie soluzioni su diversi dataset di Benchmark.
Per quanto finora detto, in questo capitolo verrano presi in considerazione:

- *Dataset UCI* [12]: il sito UCI presenta una vasta gamma di dataset ognuno dei quali è correlato ad una caratteristica ben precisa.
  Nel nostro caso, la sezione *Imbalanced Dataset* ci fornisce un'ampia collezione di dataset sbilanciati atti a testare le soluzioni concordate;

- *Dataset reale*: dataset notevolmente più complesso, ottenuto dalle rilevazioni di una compagnia assicurativa.

5.1 I dataset UCI

Il sito web UCI rappresenta una fonte di dati contrassegnati da caratteristiche ben precise (e.g. nel nostro caso verranno presi in considerazione dataset sbilanciati).
Oltre ad offrire una vasta collezione di dataset, tale repository offre la possibilità di esplorare i risultati presenti in letteratura così da confrontare l'andamento della propria analisi sperimentale con i risultati già noti.
I dataset sono disponibili nel formato *Attribute Relationship File Format (ARFF).*
Un file ARFF è un file di tipo testuale che può essere suddiviso in due sezioni: una prima di intestazione, seguita da una seconda parte contenente i dati.
La parte d'intestazione è composta da diversi *tag* nella forma *@yyyyyy*, dove *yyyyyy* indica il nome del tag.
Possibili tag ARFF possono essere:

- ***@relation:*** indica il nome del dataset. Il tag è posto come prima riga del file ARFF;

- ***@attribute:*** indica il nome dell'attributo, congiuntamente al tipo e ai valori che esso potrà assumere;

- ***@data:*** indica l'inizio delle entry del dataset.

Di seguito un esempio di dataset di tipo ARFF:

```
@relation cancer
@attribute Clump_Thickness integer [1,10]
```

@attribute Cell_Size_Uniformity integer [1,10]
@attribute Cel_Shape_Uniformity integer [1,10]
@attribute Class {benign, malignant}
@data
5,1,1,benign
5,4,4,benign
8,10,10,malignant
1,1,1,benign
5,3,3,malignant

Il Dataset *cancer* è formato da tre attributi di tipo intero, che possono assumere i valori nel range [1-10].
Il quarto attributo identifica l'etichetta di classe, che può assumere i valori *benign* o *malignant*.
Al tag *@data*, seguono le cinque entry del dataset.

5.1.1 Dataset UCI in esame

La Tabella 5.1 elenca i dataset sbilanciati che saranno utilizzati in fase di testing.
Per ognuno di essi viene indicato:

- *Dataset*: nome del dataset in esame;
- *#Classe Maj*: numero di entry appartenenti alla classe di maggioranza;
- *#Classe Min*: numero di entry appartenenti alla classe minoritaria;
- *Totale*: numero totale di entry;
- *Attributi*: numero di attributi che compongono il dataset;
- *IR*: Imbalanced Ratio del dataset in esame.

Tabella 5.1: Dataset UCI in esame

Dataset	**#Classe Maj**	**#Class Min**	**#Totale**	**#Attributi**	**IR**
vehicle0	647	199	78357	19	3
vehicle3	634	212	80468	19	2
yeast3	1321	163	91212	9	8
kr-vs-k-one_vs_fifteen	2166	78	62832	7	27
kr-vs-k-zero-one_vs_draw	2796	105	81228	7	26
shuttle-2_vs_5	3267	49	138482	10	66
car-vgood	1663	65	69984	7	25
glass6	185	29	19549	10	6
shuttle-6_vs_2-3	220	10	9820	10	22
ecoli3	301	35	18982	8	8
ecoli1	258	78	19500	8	3
new-thyroid1	180	35	8499	6	5
haberman	225	81	7691	4	2
kr-vs-k-zero_vs_fifteen	2166	27	61404	7	80
kr-vs-k-zero_vs_eight	1433	27	40880	7	53
car-good	1659	69	69984	7	24
kr-vs-k-three_vs_eleven	2854	81	82180	7	35

Tutti i dataset in esame sono accumunati dall'avere un attributo di classe binario (e.g. etichette *Positivo* e *Negativo*).
Anche se l'analisi sarà limitata al caso binario, essa è facilmente estendibile al caso in cui l'etichetta di classe assuma più di due valori.
Tale scelta implementativa di limitarsi al solo caso binario nasce dall'esigenza di semplificare il problema che già risulta notevolmente complesso nel suo caso più semplice.
Ulteriore caratteristica comune alla gran parte dei valori degli attributi è essere caratterizzati da valori di tipo continuo.
Questo ha richiesto in alcuni casi una fase di preprocessing per la trasformazione dei valori continui in valori discreti grazie ad un lavoro di discretizzazione.
Gli attributi che di per sé presentavano già dei valori discreti o corrispondevano ad etichette di classe ben precise sono rimasti inalterati.
Si noti che la forza del formato *.arff* è avere nel tag *@attribute* un'indicazione preliminare del formato del campo che aiuta notevolmente la fase di discretizzazione, in quanto permette di individuare facilmente i valori continui.
La difficoltà durante la fase di classificazione sarà influenzata da tre fattori:

1. Valore del fattore IR: tanto più sarà alto, tanto più sarà complessa la fase di classificazione in quanto si ha un avanzamento del fenomeno dell'overfitting;

2. Numero di attributi: un elevato numero di attributi può complicare notevolmente il problema poiché ne aumenta la complessità;

3. Valori assunti dagli attributi: sarà diverso classificare un dataset che presenta come valori degli attributi solamente valori binari, piuttosto che collezioni di dati

che contengono una grande varietà di valori.
Chiaramente la complessità aumenterà all'aumentare del numero dei possibili valori.

5.2 Un caso di studio reale

Il dataset relativo al caso di studio reale appartiene al mondo assicurativo ed è composto dalle informazioni provenienti dalle scatole nere presenti all'interno dei veicoli degli assicurati, inerenti allo stile di guida del conducente.
Con informazione qui si intende l'utilizzo di dati inerenti alla polizza stipulata, percorsi compiuti dal veicolo, stile di guida del conducente e sinistro con dei dettagli sulla sua gravità.
Il dataset si compone di un totale di 164 attributi: tuttavia solamente una parte di questi vengono considerati rilevanti ai fini dell'analisi.
Per tal motivo, alla classificazione, precede una fase di *feature selection*.
Per *feature selection* si intende l'estrazione degli attributi ritenuti fondamentali ai fini dell'analisi.
Ciò consente di aumentare le performance degli algoritmi di Machine Learning, in quanto essi lavoreranno solamente sugli attributi rilevanti.
Tale selezione è stata effettuata congiuntamente all'esperto di dominio: il risultato è stato una riduzione del numero di attributi da 164 a 77.
Tra gli altri, sono stati riscontrati quattro attributi di grande rilevanza: gli indicatori del livello di rischio associato ad una polizza.
Questi ultimi sono fondamentali poiché costituiranno l'etichetta di classe nella fase di classificazione.
Non si è interessati a conoscerne il significato, nel seguito si parlerà in modo generico di Classe A, B, C e D.
L'esperto di dominio, sarà interessato a valutare se una polizza sarà o meno associata ad uno dei quattro livelli di rischio (classificazione booleana, SI o NO $\implies$ rischio ALTO o BASSO).
I dati provenienti dalle scatole nere vengono denominati *dati grezzi*.
Per *dato grezzo* si intende il dato originale, nato dalla fase d'acquisizione, su cui non è stata effettuata alcuna fase di preprocessing.
A seguito del lavoro di selezione delle features principali, segue la fase di preprocessing.
Inizialmente, i dati vengono ripuliti dal rumore e vengono risolte eventuali problematiche legate alla bontà del dato (e.g. valori mancanti).
Successivamente, i valori vengono trasformati da continui a discreti per mezzo del processo di discretizzazione.
Ultimata la fase di preprocessing, si passa alla fase di *trasformazione dei dati*.
I dati grezzi hanno la caratteristica di essere misurati per un certo intervallo temporale: vengono riportate, a titolo d'esempio, due possibili entry di un contesto monetario che subiscono l'aggregazione:

Tabella 5.2: Esempio Dati Grezzi

Data Inizio Misurazione	**Data Fine Misurazione**	**Ricavi**	**Perdite**
2018-12-30 10:10:00	2018-12-30 10:20:00	20.3	36.3
2018-12-30 10:20:00	2018-12-30 10:30:00	10.4	25.2

La fase di *trasformazione* consiste nell'aggregazione del dato secondo le granularità giornaliera, mensile ed annuale.
Un'aggregazione del dato offre un'applicazione ottimale degli algoritmi di Machine Learning.
Il lavoro di trasformazione del dato è stato oggetto di un altro lavoro di tesi, per tal motivo in questa sede non vi è interesse a capire come si è passati dal dato grezzo, ad una granularità giornaliera ed annuale.
Qui si dà importanza alla conoscenza del dataset annuale e alle sue caratteristiche, come esso nasca non è di nostro interesse.

A seguito delle fasi sopra riportate, si disporrà di tre diverse forme di dataset, distinte per granularità d'aggregazione:

- *Dataset con granularità giornaliera*: i dati sono aggregati giornalmente.
  In questo caso verranno riportate le informazioni sullo stile di guida e sull'utente con granularità giornaliera;

- *Dataset con granularità mensile*: i dati sono aggregati mensilmente.
  In questo caso si perde l'informazione giornaliera, poiché le informazioni verranno presentate su base mensile;

- *Dataset con granularità annuale*: i dati sono aggregati annualmente.
  Poiché la compagnia assicurativa fornisce i tracciati relativi all'anno *yyyy*, ciò significa che il dataset (e.g. dataset del 2016) conterrà una sola riga per ogni utente.

5.2.1 Caratteristiche del Dataset

A seguito delle fasi spiegate nella Sezione 5.2, il dataset con aggregazione annuale oggetto di studio si compone di 77 attributi e 82353 righe, dove ognuna di esse è riferita ad una polizza assicurativa di un cliente.
A seconda del livello di rischio utilizzato in fase di classificazione, si dispone della seguente distribuzione di dati:

Tabella 5.3: Distribuzione dei dati

Etichetta di Classe	**Classe Maggioritaria**	**Classe Minoritaria**	**IR**
Classe A	82052	1426	57
Classe B	82353	1125	73
Classe C	83177	301	276
Classe D	83055	423	196

La misura *IR* fornisce un'importantissima indicazione sulla difficoltà riscontrata in fase di classificazione.
Assegnare l'etichetta di classe C o D ad una polizza sarà estremamente complicato, in quanto non solo si dispone di una quantità di dati più grande della classe maggioritaria rispetto alla classe minoritaria, ma non si ha neanche una quantità di dati sufficienti per creare un buon modello di classificazione.

6 Risultati Sperimentali

Questo capitolo elenca i risultati ottenuti a seguito dell'applicazione delle soluzioni spiegate nelle pagine precedenti.
Possono essere individuate, a seguito della fase di ricerca, tre macro-categorie:

1. Test degli algoritmi proposti su di un caso studio reale notevolmente complesso, dove tale complessità è data dal numero di attributi e dal valore del fattore IR (Sezione 6.3);

2. Applicazione degli algoritmi sviluppati in Python sui dataset di Benchmark UCI così da poterne testare le potenzialità ed il corretto funzionamento (Sezione 6.4);

3. Ottimizzazione del classificatore associativo L^3 tramite l'approccio Cost-Sensitive per migliorarne le prestazioni su dataset di tipo sbilanciato (Sezione 6.6).

Tutti gli esperimenti sono stati effettuati su MacBook Pro (anno 2017) dotato di processore Intel Core i5 3.1 GHz, 8 GB di RAM LPDDR3 e software OS Mojave.
A seconda della macro-categoria, si possono distinguere due ambienti di sviluppo software:

- Per le categorie 1 e 2 è stato impiegato Pycharm associato all'interpreter Python v3.7.
  Per la totalità dello sviluppo in Python è stato deciso l'utilizzo della libreria Scikit Learn (vedi Sezione 4.1).
  Tale scelta è dettata dalla vastità di algoritmi di Machine Learning offerti e dalla facilità d'utilizzo;

- Per la categoria 3 è stato adoperato l'IDE Eclipse a cui è stata associata la versione 9 di Java.

Nella totalità dei casi riguardanti le categorie 2 e 3 si è utilizzata la *Stratified Cross Validation* con *K-Fold = 5*.
La categoria 1 ingloba una validazione di tipo *Stratified Cross Validation* e *Leave One Out Validation*, a seconda del tipo di approccio usato (Oversampling nel primo caso, Undersampling nell'altro).

Data la tipologia di analisi effettuata, è stato deciso di validare i propri risultati in termini di Precision, Recall e F1-Score della classe minoritaria, escludendo dalla valutazione la misura di accuratezza.
Per capire le ragioni di tale scelta, supponiamo di ottenere la seguente matrice di confusione (Tabella 6.1) a seguito di una fase di classificazione:

Tabella 6.1: Risultato di una classificazione di un dataset sbilanciato

	Precisione	**Richiamo**	**F1-Score**	**Supporto**
MajClass	99%	88%	93%	82354
MinClass	2%	18%	4%	1125
Avg/Totale	97%	87%	92%	83479

Da una prima analisi, legata ad una visione del valor medio di Precisione, Richiamo e F1-Score, sembrerebbe che la strategia di classificazione implementata abbia raggiunto ottimi risultati.
Un'ulteriore conferma si riterrebbe essere data da un valore d'accuratezza pari al 98%.
La realtà tuttavia è ben diversa.
Notiamo infatti che:

- Si hanno valori molto alti per le misure relative alla classe maggioritaria, che fanno aumentare quello dell'andamento medio della classificazione.
  Ciò è dovuto ad un overfitting durante la fase di training verso la classe maggioritaria, che ha una presenza molto più ampia rispetto alla classe minoritaria;

- Le misure relative alla classe minoritaria hanno uno score molto basso.
  Il classificatore è riuscito a predire correttamente solamente 68 elementi su un totale di 1125;

- Il valore dell'accuratezza risulta molto elevato poiché ingloba dentro di sé anche la classe maggioritaria.
  Su un totale di 83479 entry ne sono state predette correttamente ben 82178, il che sembrerebbe un numero altissimo.
  In realtà, come detto nel punto precedente, sono solamente 68 gli elementi della classe minoritaria predetti correttamente.

6.1 L'obbiettivo dell'analisi

Tutte le problematiche affrontate hanno come nucleo comune la classificazione di dataset di tipo sbilanciato ma, a seconda del contesto applicativo, possono essere distinti scopi molteplici.
Di seguito si propone un elenco dei diversi obbiettivi perseguiti in fase di ricerca, suddivisi per sezione di presentazione:

- Sezione 6.3 - Risultati caso di studio insurance: sfruttando quanto esposto nelle Sezioni 4.3 e 4.4 si vedrà come l'applicazione delle varie tecniche migliori di volta in volta il risultato atteso.
  Si dimostrerà che, data la complessità e la quantità dei dati presenti all'interno della collezione di dati, un approccio di tipo Undersampling sarà predominante rispetto agli altri.

- Sezione 6.4 - Risultati dataset di Benchmark: si vuole dimostrare che le soluzioni proposte, per particolari tipologie d'algoritmo, migliorano quanto esistente in letteratura.
  In particolare, si vorranno mettere in risalto le potenzialità degli approcci di tipo SMOTE rispetto a quelli basati su Clustering.
  A tal proposito, si proverà che il Clustering migliorerà la semplice classificazione, ma tuttavia non indurrà ad un risultato ottimale.
  Al contrario una proposta di tipo SMOTE Oversampling, oltre a migliorare la semplice classificazione, produrrà delle risposte migliori in termini di F1-Score.
  Per ultimo, si vedrà che il classico SMOTE potrà esser migliorato ulteriormente tramite l'utilizzo delle due tecniche di valutazione e rimozione.

- Sezione 6.6 - Risultati L^3 Cost Sensitive: il classificatore L^3 di per sé fornisce un'ottima implementazione d'algoritmo di tipo associativo.
  Lo scopo sarà migliorare l'esistente implementazione attraverso l'inserimento di un approccio di tipo Cost-Sensitive, dimostrando che si otterranno risultati migliori in termini di F1-Score sfruttando la matrice di costo.

6.2 Presentazione dei risultati

Come già detto nella sezione introduttiva del Capitolo 6, la discussione degli indici di qualità per la classe maggioritaria assume poco significato.
Ai fini della presentazione dei risultati sperimentali relativi ai dataset UCI (si fa riferimento alla Tabella 5.1 per conoscerne la composizione e la complessità), le conclusioni verranno presentate in una matrice così composta:

- Le righe rappresentano il dataset utilizzato;
- Le colonne rappresentano la strategia utilizzata;
- Per ogni strategia, i risultati vengono esposti in termini di Precision, Recall e F1-Score;
- Le celle evidenziate in rosso rappresentano una situazione di vittoria (in termini di punti percentuali) rispetto alla soluzione già presente in letteratura;
- Le celle evidenziate in blu indicano l'uguaglianza tra valori di soluzioni differenti;
- L'ultima riga della matrice (Average) rappresenta la media sui risultati ottenuti (media effettuata su ogni colonna).

Nel caso di studio reale, verranno presentati passo dopo passo gli step seguiti in fase di validazione delle proprie soluzioni.
La parte d'analisi eseguita in unione alla società assicurativa riveste un ruolo chiave di questa tesi, pertanto si vorrà dare importanza ai vari step ed alle varie problematiche riscontrate in fase di sviluppo.
La validazione dei dataset di Benchmark UCI ricopre un altro step fondamentale, nato

dalla necessità di portare avanti una fase di ricerca atta a definire delle possibili soluzioni al problema dei dataset sbilanciati.

6.3 Risultati caso di studio insurance

Nella Sezione 5.2 sono stati spiegati i diversi livelli d'aggregazione disponibili per il dataset delle polizze assicurative.
Di seguito sono presentate due opzioni percorribili a partire dalla tipologia di dato a nostra disposizione:

- *Analisi a breve termine*: vengono utilizzati i dataset giornalieri e mensili.
  Con questo tipo d'analisi si intende rispondere a domande del tipo: *Sarà possibile predire, con una certa accuratezza, se il cliente avrà un incidente nel mese seguente a quello della predizione? Fissando una certa finestra temporale per il training sarà possibile prevedere cosa succederà nella settimana seguente?*

- *Analisi a lungo termine*: viene utilizzato il dataset con aggregazione annuale.
  In tal caso si intende rispondere a domande del tipo: *Sarà possibile, a partire dall'anno 2016, prevedere se l'assicurato avrà un incidente o meno nel prossimo anno?*

Di nostro interesse è stata un'analisi a lungo termine, dove l'obbiettivo prefissato era quello di predire, con la miglior accuratezza possibile, i valori dei quattro livelli di rischio nell'anno seguente rispetto all'anno del dataset su cui è stato costruito il modello.
Si vuol precisare quindi che la finalità di questa seconda parte d'analisi si differenzia totalmente da ciò che si è perseguito nella Sezione 6.4: in questo caso ci interesserà trovare la configurazione che porti al miglior risultato possibile.
Questo è dettato dalla necessità di supportare la compagnia assicurativa nell'analisi delle polizze potenzialmente rischiose.
Di seguito vengono riportati i risultati sperimentali e un commento ad essi per ogni tipologia d'algoritmo testato.

6.3.1 Il primo tentativo di classificazione

Il primo tentativo di analisi consiste nell'applicazione di vari algoritmi di classificazione, senza l'utilizzo di particolari strategie.
Poiché gli algoritmi del package Scikit-Learn non sono in grado di lavorare con stringhe, sarà necessario utilizzare sia per la fase di training che per la fase di testing solamente input numerici.
Inoltre, poiché il nostro interesse sarà predire solamente uno dei quattro indici di rischio, escludendo durante la classificazione di uno di essi gli altri tre, il dataset viene opportunamente processato al fine di permetterne l'analisi:

1. *Label Encoding*: consiste nell'associare ad ogni stringa un numero.
   A seguito della discretizzazione, i possibili valori assunti dai vari attributi saranno

del tipo *Range-n*, dove n rappresenta l'intervallo d'appartenenza.
Tale operazione è assolutamente trasparente all'utente, è solo un artificio utile per permettere la classificazione.

2. *Esclusione degli indici di rischio:* viene tenuto all'interno del training set solamente l'indice di rischio con il quale si vorrà etichettare il dataset di test.
Ciò è necessario poiché la conoscenza degli altri indici non è garantita nel test set.
Si precisa che il formato di training dovrà essere il medesimo del formato di test.

3. *Rimozione della chiave primaria*: l'ID della polizza, rappresenta solamente l'identificativo della polizza assicurativa a cui è associato un cliente e non influisce in alcun modo sulla valutazione del livello di rischio.
Per tal motivo l'attributo è stato rimosso in fase di classificazione.
Se si vuol conoscere l'associazione *polizza* $\implies$ *livello di rischio*, ciò sarà possibile nella fase immediatamente successiva alla classificazione, in quanto tale processo non influisce sull'ordinamento delle entry classificate.

A seguito di quanto detto, per ogni livello di rischio sono stati ottenuti i risultati mostrati nelle Tabelle 6.2, 6.3, 6.4 e 6.5.
Tali tabelle mostrano nella parte sinistra (a) la matrice di confusione così da poter avere un'idea sul numero di TP, TN, FP e FN, mentre nella parte destra (b) sono mostrati i risultati delle misure di Precisione, Richiamo e F1-Score.
L'etichetta *NO* rappresenta la classe maggioritaria ed è associata ad un rischio basso (Rischio Alto: NO), mentre la classe *SI* rappresenta la classe minoritaria (Rischio Alto: SI):

Tabella 6.2: Risultati della classificazione per la Classe A

(a) Matrice di confusione

	NO	**SI**
NO	70997	11056
SI	1162	264

(b) Statistiche

	Precisione	**Richiamo**	**F1-Score**
NO	98%	87%	92%
SI	2%	19%	4%

Tabella 6.3: Risultati della classificazione per la Classe B

(a) Matrice di confusione

	NO	**SI**
NO	79872	2482
SI	270	855

(b) Statistiche

	Precisione	**Richiamo**	**F1-Score**
NO	100%	97%	98%
SI	26%	76%	38%

Tabella 6.4: Risultati della classificazione per la Classe C

(a) Matrice di confusione

	NO	**SI**
NO	80157	3021
SI	129	172

(b) Statistiche

	Precisione	**Richiamo**	**F1-Score**
NO	100%	96%	98%
SI	5%	57%	10%

Tabella 6.5: Risultati della classificazione per la Classe D

(a) Matrice di confusione

	NO	**SI**
NO	80638	2417
SI	161	262

(b) Statistiche

	Precisione	**Richiamo**	**F1-Score**
NO	100%	97%	98%
SI	10%	62%	17%

Notiamo una gran differenza tra la qualità della classificazione della classe *NO* e *SI*. Il gran numero di falsi positivi è dato dal fenomeno dell'overfitting, per il quale si tende ad assegnare l'etichetta di classe *SI*.
Questa prima parte si è posta come obbiettivo la spiegazione della gestione del dataset e l'utilizzo di Python al fine di creare un classificatore e permetterne l'utilizzo. Non si vuole in alcun modo raggiungere un buon risultato, in quanto attualmente sarebbe impossibile con i mezzi a disposizione.
Da ora in poi ci si propone di migliorare le prestazioni con specifico riferimento alla classe minoritaria.

6.3.2 L'Oversampling per il dataset insurance

Ci si aspetta che l'utilizzo dell'Oversampling porti ad un grande incremento delle performance.
In questa sezione si mostrano i risultati dell'applicazione di tale metodologia.
In particolare, si vuole avere un riscontro positivo rispetto a quanto di già ottenuto nelle Tabelle 6.2, 6.3, 6.4 e 6.5 mostrate in precedenza.
I risultati sono mostrati nelle Tabelle 6.6, 6.7, 6.8 e 6.9.

Tabella 6.6: Risultati della classificazione per la Classe A

(a) Matrice di confusione

	NO	**SI**
NO	79551	2501
SI	324	1102

(b) Statistiche

	Precisione	**Richiamo**	**F1-Score**
NO	100%	97%	98%
SI	31%	77%	44%

Tabella 6.7: Risultati della classificazione per la Classe B

(a) Matrice di confusione

	NO	**SI**
NO	79623	2730
SI	215	910

(b) Statistiche

	Precisione	**Richiamo**	**F1-Score**
NO	100%	97%	98%
SI	25%	81%	38%

Tabella 6.8: Risultati della classificazione per la Classe C

(a) Matrice di confusione

	NO	**SI**
NO	80157	3021
SI	129	172

(b) Statistiche

	Precisione	**Richiamo**	**F1-Score**
NO	100%	95%	97%
SI	5%	73%	10%

Tabella 6.9: Risultati della classificazione per la Classe D

(a) Matrice di confusione

	NO	**SI**
NO	80208	2847
SI	102	321

(b) Statistiche

	Precisione	**Richiamo**	**F1-Score**
NO	100%	97%	98%
SI	10%	76%	18%

Come si può notare, anche se vi è stato un miglioramento rispetto all'utilizzo del semplice algoritmo di classificazione, ancora si è lontani da performance accettabili.
Ma come mai in questo caso non si ottengono buoni risultati?
In letteratura l'utilizzo dell'Oversampling sembrerebbe implicare un notevole incremento delle performance, in quanto un approccio di questo tipo consentirebbe di ridurre di molto l'IR, portando tale misura circa ad 1.
Ciò è possibile poiché non si ha una grande distanza tra il numero di elementi della classe maggioritaria e minoritaria.
Nel caso del dataset insurance non può esser fatto lo stesso ragionamento.
Il numero di elementi della classe maggioritaria è molto più grande rispetto a quello della classe minoritaria.
L'idea successiva sarà combinare insieme una serie di classificatori (i classificatori con richiamo più alto) ed utilizzare una strategia a voto di maggioranza al fine di incrementare le performance.

6.3.3 Voto di maggioranza

Sulla base dei risultati ottenuti nella Sezione 6.3.2, è stato notato che il Richiamo ha un valore sempre più alto rispetto alla Precisione.
Per tal motivo si è deciso di combinare insieme le predizioni di più classificatori al fine di migliorare la performance generale.
Nello specifico:

1. Vengono scelti n classificatori differenti, con n dispari.
   Di seguito viene riportato un elenco dei classificatori utilizzati durante la fase di ricerca:

   - Random Forest;
   - Decision Tree;
   - K-NN;
   - SVM;
   - Naive Bayes.

   L'ordine d'utilizzo varia in base all'indice di rischio, in quanto ognuno di essi presenta un valore di richiamo differente.
   A titolo d'esempio, con $n = 3$, nel caso dell'indice di rischio relativo alla Classe B, verrebbero scelti Random Forest, Decision Tree ed SVM.

2. Ogni classificatore viene addestrato con il medesimo training set.
   Qui non viene quindi utilizzata alcuna strategia di Bagging in quanto il dataset rimarrà sempre il medesimo.

3. Ogni dato di test viene classificato da tutti i modelli a disposizione.
   In questo caso non viene associato alcun costo alla predizione, come fatto ad esempio con L^3: ogni risultato avrà lo stesso peso.
   Il valore finale assegnato al dato incognito corrisponderà all'etichetta di classe maggiormente predetta.

A seguito di diverse prove sperimentali, è stato deciso l'utilizzo di un valore $n = 3$.
Tale scelta è stata effettuata analizzando i risultati in termini di Richiamo e Precisione della classe minoritaria.
Il valore $n = 3$ non deve essere ritenuto casuale.
Fino al momento in cui vengono utilizzati solamente tre classificatori, si ha una combinazione dei risultati affidabile: il probabile errore di un classificatore viene compensato da una classificazione possibilmente corretta degli altri due.
Con un valore $n = 5$ non si presenta tale comportamento.
Si può presupporre che piuttosto d'avere una miglioria nel risultato, gli altri classificatori (quelli con valore di richiamo più basso) tendano a portare il risultato verso una strada errata.

Tabella 6.10: Risultati della classificazione per la Classe A

(a) Matrice di confusione

	NO	**SI**
NO	80792	1260
SI	693	733

(b) Statistiche

	Precisione	**Richiamo**	**F1-Score**
NO	99%	98%	92%
SI	37%	51%	43%

Tabella 6.11: Risultati della classificazione per la Classe B

(a) Matrice di confusione

	NO	**SI**
NO	81150	1203
SI	602	523

(b) Statistiche

	Precisione	**Richiamo**	**F1-Score**
NO	99%	99%	99%
SI	30%	46%	37%

Tabella 6.12: Risultati della classificazione per la Classe C

(a) Matrice di confusione

	NO	**SI**
NO	81846	1331
SI	206	95

(b) Statistiche

	Precisione	**Richiamo**	**F1-Score**
NO	100%	98%	99%
SI	7%	44%	11%

Tabella 6.13: Risultati della classificazione per la Classe D

(a) Matrice di confusione

	NO	**SI**
NO	81947	1108
SI	252	171

(b) Statistiche

	Precisione	**Richiamo**	**F1-Score**
NO	100%	99%	99%
SI	13%	40%	20%

Dai risultati ottenuti è possibile notare che la miglioria non si ha in termini di F1-Score della classe minoritaria, ma in termini di Precisione.

6.3.4 Un approccio differente: l'Undersampling

Anche se gradualmente, al variare delle diverse strategie, si nota un miglioramento nell'andamento delle misure, anche se si è ancora lontani da risultati che possano essere ritenuti accettabili.
Il termine *accettabile* non ha una soglia ben definita, poiché l'errore nella classificazione è fortemente correlato ad una difficoltà intrinseca legata alla struttura del dataset.
Tuttavia, non classificare correttamente almeno il 50% delle entry di test, è un grosso limite.
Sembrerebbe che ciò che impedisce di ottenere buoni risultati sia legato alla quantità di elementi della classe maggioritaria: a prescindere dal modo e dalla quantità di elementi della classe minoritaria duplicati, tale quantità non potrà mai raggiungere il numero di elementi dell'altra classe.
Questa considerazione non è vera in assoluto, ma lo è in questo caso dove abbiamo un rapporto tra le due classi di circa 80000:1000.
Per tali ragioni, in questa sezione viene presentato un approccio totalmente opposto: l'Undersampling.
In fase di sviluppo sono state provate entrambe le possibili soluzioni: *Random* e *Clustered Undersampling*.

Qui è stato deciso di mostrare, in forma integrale, solamente il risultato relativo al *Clustered Undersampling*, in quanto è la strategia che offre risultati migliori in termini di *F1-Score* della classe minoritaria.
Quanto detto non è casuale, ma è perfettamente in linea con quello che è stato ipotizzato nella Sezione 4.3.2: nel Random Undersampling, poiché i dati vengono scelti ed eliminati in maniera casuale, sarà possibile una perdita d'informazione importante.
L'utilizzo del Clustered Undersampling, permette di eliminare gli elementi di cui già si ha una conoscenza nel training set, mantenendo quelli che potrebbero essere importanti, poiché rari, in fase di training.
In questo caso, data la grande dimensione del training set, non è stato possibile utilizzare la Silhouette Measure.
Si ricorda che per la sua valutazione viene calcolato, per ogni dato p appartenente ad un cluster, la distanza tra p ed ogni altro dato clusterizzato q al fine di avere una stima del grado di coesione.
Ciò comporta una complessità $O(n^2)$.
Tale complessità è sicuramente accettabile nei dataset UCI, tuttavia un ragionamento analogo non potrà esser fatto nel caso di questo dataset contenente circa 80000 entry.
Per tal motivo, è stata svolta una ricerca esplorativa atta a testare le diverse configurazioni possibili del K-Means e del DBSCAN, facendo variare il parametro K nel primo caso e $minPts$ ed Eps nel secondo.
A seguito di un'analisi dei risultati, si è scelto l'utilizzo del K-Means con parametro $n\_cluster = 3$.

Per ottenere i seguenti risultati, è stata utilizzata una percentuale di Undersampling pari al 5%. Ciò significa che verrà mantenuto solamente il 5% del numero di elementi appartenenti alla classe maggioritaria del training set.
Come tecnica di validazione, si è optato per l'utilizzo della Leave One Out Validation.
I risultati per $n\_cluster = 3$ e $under\_perc = 5\%$ sono mostrati nelle Tabelle 6.14, 6.15, 6.16 e 6.17:

Tabella 6.14: Risultati della classificazione per la Classe A

(a) Matrice di confusione

	NO	**SI**
NO	1883	256
SI	247	1179

(b) Statistiche

	Precisione	**Richiamo**	**F1-Score**
NO	88%	88%	88%
SI	82%	83%	82%

Tabella 6.15: Risultati della classificazione per la Classe B

(a) Matrice di confusione

	NO	**SI**
NO	1496	191
SI	216	909

(b) Statistiche

	Precisione	**Richiamo**	**F1-Score**
NO	87%	89%	88%
SI	83%	81%	82%

Tabella 6.16: Risultati della classificazione per la Classe C

(a) Matrice di confusione

	NO	**SI**
NO	242	59
SI	67	234

(b) Statistiche

	Precisione	**Richiamo**	**F1-Score**
NO	78%	80%	89%
SI	78%	80%	89%

Tabella 6.17: Risultati della classificazione per la Classe D

(a) Matrice di confusione

	NO	**SI**
NO	524	110
SI	99	324

(b) Statistiche

	Precisione	**Richiamo**	**F1-Score**
NO	84%	83%	83%
SI	75%	77%	76%

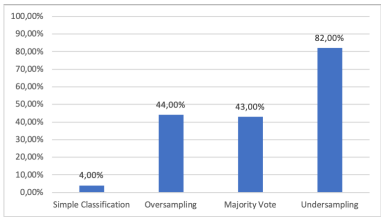
Come si può notare, i risultati di questo approccio sono nettamente migliori.
Tuttavia è doveroso riflettere sul seguente concetto: nel caso della Cross Validation viene presa in considerazione per il testing una grande parte del dataset (la grandezza dipenderà dal numero di fold utilizzati).
Questo significa sottrarre informazione alla fase di training, poiché il dataset utilizzato per il testing non verrà utilizzato per costruire il modello.
La Leave One Out invece valuta solamente un'entry del dataset: la parte restante (dataset - elemento di test) verrà utilizzata interamente per effettuare il training.
Perché sarà impossibile effettuare l'Undersampling associato alla Cross Validation?
Supponiamo di voler valutare le prestazioni nella classificazione della classe C. Ad esso sono associate 301 entry appartenenti alla classe minoritaria.
Immaginando di voler applicare la Stratified Cross Validation con $n\_split = 5$, verranno creati 5 fold differenti ognuno contenente 60 elementi della classe minoritaria. Notiamo tuttavia che 60 elementi sono un numero troppo esiguo per poter addestrare un classificatore.
Da ciò nasce la necessità di dover applicare una strategia di validazione differente.

6.3.5 Conclusioni al caso di studio reale

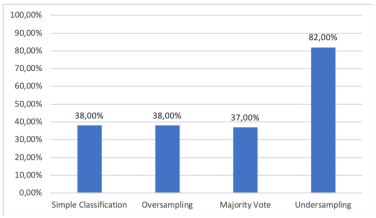
Nelle sezioni precedenti sono state proposte diverse metodologie per la gestione dell'imbalanced dataset relativo al caso di studio insurance.
Una parte della fase sperimentale di questa tesi, è associata al tentativo di migliorare le performance nella classificazione dei quattro indici di rischio (Classe A, Classe B, Classe C e Classe D) definiti con l'esperto di dominio.
A seguito di diversi tentativi, sono state proposte tre strategie: due a livello di dato (come l'Undersampling e l'Oversampling) ed una a livello d'algoritmo (Majority Voting).
A scopo riepilogativo, viene mostrato nella Figura 6.1 l'andamento dei progressi di ricerca.

Figura 6.1: Risultati Finali

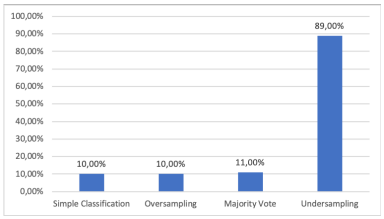
(a) F1-Score Classe A



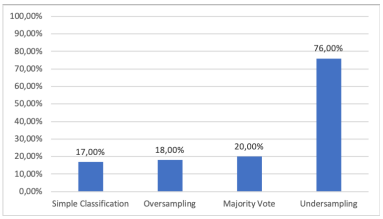
(b) F1-Score Classe B



(c) F1-Score Classe C



(d) F1-Score Classe D



Di seguito viene riportata la miglior configurazione individuata in fase di ricerca, con la quale si è ottenuto il più alto valore di F1-Score per la classe minoritaria:

- Algoritmo di classificazione: Random Forest;
- Strategia: Clustered Undersampling;
- Algoritmo di Clustering: K-Means con $n\_cluster = 3$;
- Percentuale di Undersampling: 5%;
- Strategia di Validazione: Leave One Out Validation.

6.4 Risultati dataset di Benchmark

L'obbiettivo di questa prima fase d'analisi sarà quello di validare le proprie soluzioni e porle a confronto con quanto di già esistente in letteratura.
Per la validazione dei propri risultati sono stati utilizzati diversi algoritmi di classificazione appartenenti al package Scikit Learn (v 20.0.0).
In tal senso vengono prese in analisi le seguenti strategie:

- *Classificazione Semplice*: si utilizza il classificatore senza applicare nessuna tecnica d'incremento delle prestazioni;
- *Classificazione basata su singolo dato di test*: implementata attraverso gli algoritmi di clustering K-Means e DBSCAN;
- *Classificazione basata su cluster come dato di test*: implementazione basata su Clustering attraverso K-Means e DBSCAN;
- *SMOTE Oversampling*: viene utilizzata l'implementazione dello SMOTE reperibile all'interno del package imblearn.over_sampling.SMOTE [23];

- *SMOTE con valutazione degli elementi della classe minoritaria*: a seguito dell'applicazione dello SMOTE, gli elementi sintetizzati come appartenenti alla classe minoritaria, che in realtà non ne fanno parte, vengono eliminati dal training set;
- *SMOTE con valutazione degli elementi della classe maggioritaria e minoritaria*: dopo aver applicato lo SMOTE, i dati che partecipano ad una classificazione errata sono eliminati dal training set.

Non è presente l'utilizzo dell'Undersampling a causa della struttura dei dataset UCI.
Come già spiegato in linea teorica, l'utilizzo dell'Undersampling è adatto quando vi è una grande disparità tra numero di elementi della classe maggioritaria e minoritaria unita ad un elevato numero di record che compongono il dataset.
Questo consente di escludere dalla creazione del modello le entry della classe maggioritaria mantenendo comunque un numero di record elevato adatto alla creazione di un modello di classificazione.
Per dataset relativamente piccoli, questo approccio non è utilizzabile, in quanto ridurrebbe notevolmente la sua dimensione, fintanto da non essere più utilizzabile per la classificazione.

6.4.1 Validazione tecniche allo stato dell'arte

In questa sezione si vuol mettere in evidenza la capacità delle tecniche di clustering di migliorare le performance della semplice classificazione.
A tal proposito, nella Tabella 6.18 vengono messi a confronto i risultati ottenuti a seguito della normale classificazione con i valori relativi all'approccio basato su Clustering con entry e con cluster come dato di test.

Tabella 6.18: Risultati

	Classificazione semplice			Dato come test			Cluster come test		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
kr-vs-k-zero-one_vs_draw	44%	**81%**	56%	**61%**	62%	**61%**	59%	52%	55%
new-thyroid1	0%	0%	0%	**28%**	**66%**	**39%**	27%	63%	38%
vehicle3	**52%**	24%	33%	41%	**48%**	**44%**	41%	47%	**44%**
kr-vs-k-zero_vs_eight	11%	41%	17%	**24%**	**63%**	**35%**	14%	33%	20%
kr-vs-k-zero_vs_fifteen	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
vehicle0	78%	**88%**	83%	**87%**	83%	**85%**	**87%**	83%	**85%**
ecoli1	**85%**	14%	24%	47%	78%	**59%**	39%	**94%**	55%
kr-vs-k-one_vs_fifteen	0%	0%	0%	**3%**	19%	5%	**3%**	**79%**	**6%**
kr-vs-k-three_vs_eleven	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**
ecoli3	0%	0%	0%	**50%**	**43%**	**46%**	**50%**	28%	37%
haberman	**62%**	25%	36%	37%	**62%**	**46%**	37%	57%	45%
glass6	57%	**76%**	66%	**91%**	72%	**80%**	62%	45%	52%
shuttle-2_vs_5	96%	98%	97%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
car-vgood	13%	20%	16%	**18%**	31%	**23%**	10%	**37%**	16%
yeast3	0%	0%	0%	**71%**	**51%**	**59%**	44%	47%	45%
shuttle-6_vs_2-3	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
car-good	0%	0%	0%	7%	**17%**	**10%**	**9%**	10%	9%
Average	41%	39%	36%	**50%**	**58%**	**52%**	46%	57%	47%

L'algoritmo utilizzato per la validazione delle strategie elencate nella Tabella 6.18 è una Neural Network, opportunamente configurato per ogni dataset così da sfruttarne le massime potenzialità.
È facile notare che la soluzione che utilizza il singolo dato come caso di test offre le prestazioni migliori: nel 100% dei casi esso supera o eguaglia la semplice classificazione. Un discorso analogo, seppur con una qualità leggermente inferiore, può esser fatto per l'algoritmo che utilizza il cluster come dato di test: nel 90% dei casi vince rispetto alle tecniche standard, tuttavia in media le prestazioni della metodologia con cluster come dato di test risultano inferiori a quelle dell'approccio che utilizza un singolo dato come test.
Ciò può esser attribuito alla struttura software dell'algoritmo: la metodologia che utilizza il singolo elemento come dato di test tende ad associare quest'ultimo al miglior modello.
Questo comporta, a rigor di logica, dei risultati migliori rispetto all'associazione di un intero gruppo al modello migliore.
A tal proposito, si può notare che quest'ultimo vince sulle altre tre tecniche in termini di Precision, Recall ed F1-Score.
Valutando la vittoria nel suo valore più importante, ovvero secondo l'F1-Score, la metodologia che utilizza il singolo elemento come dato di test ne è caratterizzata per un valore medio pari a 52%.
Il dato, seppur non ancora soddisfacente, risulta notevolmente positivo in termini di miglioramento rispetto alla tecnica base.

Tutto ciò implica che il Clustering effettivamente aiuta la classificazione nella sua forma più semplice.
Poiché questi risultati non possono esser ritenuti soddisfacenti in termini di performance complessive, nella sezione successiva verranno proposte altre soluzioni che li miglioreranno ulteriormente.

6.4.2 Validazione proposte di ricerca

Di seguito verranno presentati i risultati sperimentali relativi alla Sezione 4.4.
Come per la validazione delle tecniche allo stato dell'arte, sono stati utilizzati i dataset di Benchmark UCI.
Tuttavia in questo caso, per dimostrare la forza della propria soluzione, i risultati verranno presentati come segue: lo SMOTE con valutazione della classe minoritaria e di tutto il dataset saranno validati su diversi algoritmi di classificazione, i cui risultati sono mostrati nelle Tabelle 6.19, 6.20, 6.21, 6.22, 6.23 e 6.24.
A seguito di ogni tabella, sarà presentato un piccolo sommario che indica, per ogni tecnica di validazione, il numero di casi in cui l'algoritmo è stato ritenuto vincente (dove per vincente si intende la tecnica con il valore più alto di Precision, Recall o F1-Score).
Questa tipologia di analisi è dettata da una ragione ben precisa: si vogliono capire vantaggi e svantaggi della propria implementazione, cercando di scindere i casi per cui l'utilizzo dello SMOTE con valutazione e rimozione è consigliabile da quelli in cui il suo utilizzo causa un degrado nelle prestazioni del classificatore.
In linea alla necessità di generalizzare i propri test, questi verranno effettuati sui seguenti modelli di classificazione:

- MLP: implementazione di una Rete Neurale di tipo Multi Layer Perceptron;
- SVC: algoritmo corrispondente ad una macchina a vettori di supporto;
- Gradient Boosting: algoritmo di Ensemble che aumenta la forza degli Alberi di Decisione;
- KNN: implementazione del k-nearest neighbors;
- Decision Tree: classificatore basato su un modello ad Albero di Decisione;
- GaussianNB: classificatore basato sul teorema di Bayes con ipotesi Naïve.

La scelta di questi modelli non è casuale, in quanto ognuno di essi si differenzia dall'altro in maniera sostanziale: le SVM, NN e KNN utilizzano (anche se in maniera differente) un modello matematico costruito in uno spazio n-dimensionale per effettuare la classificazione, gli Alberi di Decisione creano delle scelte basate sul valore di un attributo ed infine il GaussianNB è supportato dal teorema di Bayes.

Tabella 6.19: Risultati MLP

	SMOTE			Valutazione classe minoritaria			Valutazione dataset di training		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
kr-vs-k-zero-one_vs_draw	**83%**	81%	82%	82%	**96%**	**88%**	74%	**96%**	84%
new-thyroid1	86%	91%	88%	**89%**	**94%**	**91%**	**89%**	**94%**	**91%**
vehicle3	**48%**	32%	38%	45%	**45%**	**45%**	46%	34%	39%
kr-vs-k-zero_vs_eight	21%	**70%**	32%	**31%**	**70%**	**43%**	19%	67%	30%
kr-vs-k-zero_vs_fifteen	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
vehicle0	**85%**	91%	88%	**85%**	94%	**89%**	80%	**97%**	88%
ecoli1	64%	79%	71%	**71%**	78%	**74%**	62%	**82%**	71%
kr-vs-k-one_vs_fifteen	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
kr-vs-k-three_vs_eleven	51%	**94%**	66%	**85%**	**94%**	**89%**	68%	**94%**	79%
ecoli3	10%	**60%**	17%	**64%**	20%	**30%**	10%	**60%**	17%
haberman	40%	60%	48%	**48%**	36%	41%	41%	**62%**	**49%**
glass6	70%	79%	74%	**77%**	79%	**78%**	69%	**83%**	75%
shuttle-2_vs_5	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
car-vgood	38%	60%	47%	**46%**	**68%**	**55%**	**46%**	**68%**	**55%**
yeast3	67%	87%	76%	**73%**	82%	**77%**	65%	**88%**	75%
shuttle-6_vs_2-3	90%	**90%**	90%	**100%**	**90%**	**95%**	**100%**	**90%**	**95%**
car-good	7%	51%	12%	**28%**	25%	**26%**	4%	**72%**	8%
Average	62%	77%	66%	**72%**	74%	**71%**	63%	**81%**	68%

L'algoritmo *MLP* presenta le seguenti caratteristiche:

- SMOTE:
  - ◇ Precision: 2 casi di vittoria, 4 di pareggio e 11 di sconfitta;
  - ◇ Recall: 0 casi di vittoria, 7 di pareggio e 10 di sconfitta;
  - ◇ F1-Score: 0 casi di vittoria, 3 di pareggio e 14 di sconfitta.
- SMOTE con valutazione della classe minoritaria:
  - ◇ Precision: 8 casi di vittoria, 7 di pareggio e 2 di sconfitta;
  - ◇ Recall: 1 caso di vittoria, 9 di pareggio e 7 di sconfitta;
  - ◇ F1-Score: 10 casi di vittoria, 6 di pareggio e 1 di sconfitta.
- SMOTE con valutazione del dataset di training:
  - ◇ Precision: 0 casi di vittoria, 6 di pareggio e 11 di sconfitta;
  - ◇ Recall: 6 casi di vittoria, 9 di pareggio e 2 di sconfitta;
  - ◇ F1-Score: 1 caso di vittoria, 6 di pareggio e 10 di sconfitta.

Analizzando i risultati in termini di F1-Score, si evidenzia che la strategia SMOTE con valutazione e rimozione della classe minoritaria risulta vincitrice 10 volte su 17.
Inoltre, si contano 6 situazioni di pareggio e solamente un caso in cui la strategia proposta non migliora il classico approccio SMOTE (dataset haberman).

La strategia SMOTE con valutazione e rimozione del dataset di training non ottiene i risultati sperati: si evidenziano 6 situazioni di pareggio e solamente un caso di vittoria. È importante notare che non si ha mai un degrado delle prestazioni evidente rispetto allo SMOTE, nel caso peggiore si rileva una situazione di pareggio.
Sono da sottolineare i casi dei dataset kr-vs-k-zero-one_vs_draw, vehicle3, kr-vs-k-zero_-vs_eight, ecoli3 e car-good dove si ha un aumento delle prestazioni tra il 6% e il 13%.
Di grande rilevanza è il caso del dataset kr-vs-k-three_vs_eleven, dove si ha un notevole incremento (23%).

Tabella 6.20: Risultati SVC

	SMOTE			**Valutazione classe minoritaria**			**Valutazione dataset di training**		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
kr-vs-k-zero-one_vs_draw	42%	**95%**	57%	**78%**	91%	**84%**	42%	**95%**	57%
new-thyroid1	85%	**94%**	89%	**89%**	91%	**90%**	85%	**94%**	89%
vehicle3	46%	**42%**	**44%**	**47%**	**42%**	**44%**	46%	**42%**	**44%**
kr-vs-k-zero_vs_eight	27%	78%	40%	**31%**	78%	**44%**	30%	**81%**	**44%**
kr-vs-k-zero_vs_fifteen	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
vehicle0	95%	97%	96%	**96%**	**98%**	**97%**	95%	**98%**	96%
ecoli1	67%	**83%**	**74%**	**68%**	82%	**74%**	66%	82%	73%
kr-vs-k-one_vs_fifteen	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
kr-vs-k-three_vs_eleven	67%	**94%**	78%	**96%**	**94%**	**95%**	62%	**94%**	75%
ecoli3	48%	**91%**	63%	**65%**	86%	**74%**	47%	89%	62%
haberman	26%	28%	27%	**33%**	**31%**	**32%**	22%	27%	24%
glass6	67%	**76%**	71%	**100%**	69%	**82%**	66%	72%	69%
shuttle-2_vs_5	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
car-vgood	**45%**	86%	59%	**45%**	88%	**60%**	44%	**89%**	59%
yeast3	64%	89%	74%	**80%**	82%	**81%**	64%	**91%**	75%
shuttle-6_vs_2-3	**83%**	**100%**	**91%**	**83%**	**100%**	**91%**	**83%**	**100%**	**91%**
car-good	31%	74%	44%	**37%**	**83%**	**51%**	34%	72%	46%
Average	64%	**83%**	71%	**73%**	**83%**	**76%**	63%	**83%**	70%

L'algoritmo *SVC* presenta le seguenti caratteristiche:

- SMOTE:
  - ◇ Precision: 0 casi di vittoria, 5 di pareggio e 12 di sconfitta;
  - ◇ Recall: 3 casi di vittoria, 8 di pareggio e 6 di sconfitta;
  - ◇ F1-Score: 0 casi di vittoria, 6 di pareggio e 11 di sconfitta.
- SMOTE con valutazione della classe minoritaria:
  - ◇ Precision: 12 casi di vittoria, 5 di pareggio e 0 di sconfitta;
  - ◇ Recall: 2 casi di vittoria, 7 di pareggio e 8 di sconfitta;
  - ◇ F1-Score: 10 casi di vittoria, 7 di pareggio e 0 di sconfitta.

- SMOTE con valutazione del dataset di training:
  - ◇ Precision: 0 casi di vittoria, 4 di pareggio e 13 di sconfitta;
  - ◇ Recall: 3 casi di vittoria, 9 di pareggio e 5 di sconfitta;
  - ◇ F1-Score: 0 casi di vittoria, 6 di pareggio e 11 di sconfitta.

Si evidenziano per lo SMOTE con valutazione e rimozione della classe minoritaria ben 10 casi di vittoria e 7 di pareggio.
Tali risultati possono essere letti con un'ottica positiva: si ha nel 100% dei casi il superamento o il pareggio dell'F1-Score del semplice approccio SMOTE.
I casi dei dataset kr-vs-k-zero-one_vs_draw, kr-vs-k-three_vs_eleven, ecoli3, glass6 e yeast3 sono degli esempi di miglioramento notevole, nei quali si ha un incremento delle prestazioni tra il 7% e il 27%.
Le prestazioni dello SMOTE con valutazione e rimozione del dataset di training vengono valutate nuovamente in modo negativo: in questo caso non si ha nessun esempio di vittoria rispetto agli altri algoritmi.

Tabella 6.21: Risultati Gradient Boosting

	SMOTE			**Valutazione classe minoritaria**			**Valutazione dataset di training**		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
kr-vs-k-zero-one_vs_draw	73%	87%	79%	73%	**90%**	**81%**	**76%**	84%	80%
new-thyroid1	91%	91%	91%	**92%**	**94%**	**93%**	91%	91%	91%
vehicle3	51%	55%	53%	**53%**	55%	54%	**53%**	**59%**	**56%**
kr-vs-k-zero_vs_eight	27%	78%	40%	**41%**	**81%**	**54%**	27%	78%	40%
kr-vs-k-zero_vs_fifteen	**100%**	**78%**	**88%**	**100%**	**78%**	**88%**	**100%**	**78%**	**88%**
vehicle0	86%	89%	87%	**87%**	**91%**	**89%**	86%	89%	87%
ecoli1	66%	84%	74%	**68%**	**87%**	**76%**	67%	86%	75%
kr-vs-k-one_vs_fifteen	88%	**92%**	90%	**100%**	**92%**	**96%**	**100%**	**92%**	**96%**
kr-vs-k-three_vs_eleven	49%	**94%**	64%	**64%**	**94%**	**76%**	**64%**	**94%**	**76%**
ecoli3	55%	66%	60%	**65%**	**69%**	**67%**	56%	**69%**	62%
haberman	39%	56%	46%	**52%**	32%	40%	40%	**63%**	**49%**
glass6	77%	79%	78%	**84%**	**90%**	**87%**	77%	79%	78%
shuttle-2_vs_5	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
car-vgood	39%	**75%**	51%	**50%**	**75%**	**60%**	**50%**	**75%**	**60%**
yeast3	**75%**	82%	78%	74%	**84%**	**79%**	72%	82%	77%
shuttle-6_vs_2-3	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
car-good	17%	77%	28%	**19%**	**88%**	**31%**	18%	**88%**	30%
Average	66%	81%	71%	**71%**	**82%**	**74%**	69%	**82%**	73%

L'algoritmo *Gradient Boosting* presenta le seguenti caratteristiche:

- SMOTE:
  - ◇ Precision: 1 caso di vittoria, 3 di pareggio e 13 di sconfitta;
  - ◇ Recall: 0 casi di vittoria, 6 di pareggio e 11 di sconfitta;

    - ◇ F1-Score: 0 casi di vittoria, 3 di pareggio e 14 di sconfitta.

- SMOTE con valutazione della classe minoritaria:
    - ◇ Precision: 8 casi di vittoria, 7 di pareggio e 2 di sconfitta;
    - ◇ Recall: 7 casi di vittoria, 8 di pareggio e 2 di sconfitta;
    - ◇ F1-Score: 9 casi di vittoria, 6 di pareggio e 2 di sconfitta.
- SMOTE con valutazione del dataset di training:
    - ◇ Precision: 1 caso di vittoria, 7 di pareggio e 9 di sconfitta;
    - ◇ Recall: 2 casi di vittoria, 8 di pareggio e 7 di sconfitta;
    - ◇ F1-Score: 2 casi di vittoria, 6 di pareggio e 9 di sconfitta.

Il Gradient Boosting è un altro esempio di buon funzionamento dello SMOTE con valutazione e rimozione della classe minoritaria.
Per quest'ultimo, possono essere identificate 9 vittorie e 6 casi di pareggio a seguito di un confronto con gli altri due algoritmi.
Considerando il singolo confronto con lo SMOTE, possono essere identificati 16 episodi di vittoria sui 17 casi di studio, dato che può esser letto sotto una chiave positiva in quanto si ha un superamento delle prestazioni di base del 94%.
Tra gli altri, sono rilevanti le classificazioni per i dataset kr-vs-k-zero_vs_eight, glass6 e yeast3, con un miglioramento sino al 14% in termini di punti percentuali dell'F1-Score.
Anche se migliore in termini di media sulle misure rispetto allo SMOTE, anche in questo la versione con valutazione e rimozione del dataset di training risulta meno robusto rispetto allo SMOTE con valutazione e rimozione della classe minoritaria.

Tabella 6.22: Risultati KNN

	SMOTE			Valutazione classe minoritaria			Valutazione dataset di training		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
kr-vs-k-zero-one_vs_draw	**65%**	80%	72%	**65%**	88%	**75%**	63%	**89%**	74%
new-thyroid1	77%	**94%**	85%	**82%**	**94%**	**88%**	77%	**94%**	85%
vehicle3	42%	**80%**	**55%**	**46%**	41%	43%	42%	79%	**55%**
kr-vs-k-zero_vs_eight	30%	**96%**	46%	**31%**	**96%**	**47%**	28%	**96%**	45%
kr-vs-k-zero_vs_fifteen	**90%**	**100%**	**95%**	**90%**	**100%**	**95%**	87%	**100%**	93%
vehicle0	**69%**	**100%**	**82%**	**69%**	**100%**	**82%**	68%	**100%**	81%
ecoli1	62%	84%	71%	**64%**	84%	**73%**	63%	**86%**	**73%**
kr-vs-k-one_vs_fifteen	95%	**96%**	95%	**97%**	95%	**96%**	**97%**	95%	**96%**
kr-vs-k-three_vs_eleven	**43%**	**100%**	**60%**	**43%**	**100%**	**60%**	**43%**	**100%**	**60%**
ecoli3	48%	**91%**	63%	**50%**	**91%**	**65%**	48%	**91%**	63%
haberman	39%	**53%**	45%	**53%**	41%	**46%**	36%	49%	42%
glass6	74%	**79%**	76%	**79%**	**79%**	**79%**	74%	**79%**	76%
shuttle-2_vs_5	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
car-vgood	30%	82%	44%	**33%**	**100%**	**50%**	**33%**	**100%**	**50%**
yeast3	53%	**90%**	67%	**71%**	85%	**77%**	55%	89%	68%
shuttle-6_vs_2-3	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
car-good	23%	71%	35%	26%	99%	41%	**27%**	**100%**	**43%**
Average	61%	88%	70%	**64%**	87%	**71%**	61%	**91%**	70%

L'algoritmo *KNN* presenta le seguenti caratteristiche:

- SMOTE:
  - ◇ Precision: 0 casi di vittoria, 6 di pareggio e 11 di sconfitta;
  - ◇ Recall: 4 casi di vittoria, 9 di pareggio e 4 di sconfitta;
  - ◇ F1-Score: 0 casi di vittoria, 6 di pareggio e 11 di sconfitta.
- SMOTE con valutazione della classe minoritaria:
  - ◇ Precision: 8 casi di vittoria, 8 di pareggio e 1 di sconfitta;
  - ◇ Recall: 0 casi di vittoria, 10 di pareggio e 7 di sconfitta;
  - ◇ F1-Score: 7 casi di vittoria, 8 di pareggio e 2 di sconfitta.
- SMOTE con valutazione del dataset di training:
  - ◇ Precision: 1 caso di vittoria, 5 di pareggio e 11 di sconfitta;
  - ◇ Recall: 3 casi di vittoria, 10 di pareggio e 4 di sconfitta;
  - ◇ F1-Score: 1 caso di vittoria, 7 di pareggio e 9 di sconfitta.

Si analizzano i risultati in termini di F1-Score.
Lo SMOTE con valutazione e rimozione della classe minoritaria si può ritenere vincitore in solamente 7 casi sui 17 di test, dando vita ad una situazione di parità con gli altri algoritmi per ben 8 volte.

Questi numeri non devono esser guardati solamente in maniera negativa, anche se non è presente un grande incremento delle prestazioni, si nota una decrescita del valore dell'F1-Score rispetto al semplice SMOTE in solamente un caso di test (dataset vehicle3).
In altro modo deve essere interpretata la situazione della strategia SMOTE con valutazione e rimozione del dataset di training, che non raggiunge i risultati desiderati: son presenti 7 situazioni di pareggio e solamente 1 caso di vittoria (dataset car-good).

Tabella 6.23: Risultati DecisionTree

	SMOTE			**Valutazione classe minoritaria**			**Valutazione dataset di training**		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
kr-vs-k-zero-one_vs_draw	38%	86%	53%	**67%**	67%	**67%**	38%	**90%**	53%
new-thyroid1	82%	**94%**	88%	**85%**	**94%**	**89%**	**85%**	**94%**	**89%**
vehicle3	48%	52%	50%	**50%**	**56%**	**53%**	45%	52%	48%
kr-vs-k-zero_vs_eight	22%	**81%**	35%	**100%**	63%	**77%**	22%	**81%**	35%
kr-vs-k-zero_vs_fifteen	**100%**	**78%**	**88%**	**100%**	**78%**	**88%**	**100%**	**78%**	**88%**
vehicle0	84%	**87%**	85%	**87%**	**87%**	**87%**	**87%**	**87%**	**87%**
ecoli1	62%	82%	71%	**67%**	78%	72%	64%	**84%**	**73%**
kr-vs-k-one_vs_fifteen	**86%**	**92%**	**89%**	**86%**	**92%**	**89%**	**86%**	**92%**	**89%**
kr-vs-k-three_vs_eleven	53%	**94%**	68%	**78%**	**94%**	**85%**	53%	**94%**	68%
ecoli3	44%	77%	56%	**69%**	51%	**59%**	45%	**83%**	57%
haberman	**38%**	**48%**	**42%**	31%	11%	16%	31%	36%	33%
glass6	72%	**90%**	80%	**88%**	79%	**83%**	76%	86%	81%
shuttle-2_vs_5	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
car-vgood	**23%**	**60%**	**33%**	14%	20%	16%	22%	**60%**	32%
yeast3	68%	**87%**	76%	**74%**	**87%**	**80%**	65%	**87%**	74%
shuttle-6_vs_2-3	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
car-good	**12%**	**59%**	**20%**	0%	0%	0%	10%	55%	17%
Average	60%	**80%**	66%	**70%**	68%	**68%**	60%	79%	66%

L'algoritmo *DecisionTree* presenta le seguenti caratteristiche:

- SMOTE:
  - ◇ Precision: 3 casi di vittoria, 4 di pareggio e 10 di sconfitta;
  - ◇ Recall: 3 casi di vittoria, 10 di pareggio e 4 di sconfitta;
  - ◇ F1-Score: 3 casi di vittoria, 4 di pareggio e 10 di sconfitta.
- SMOTE con valutazione della classe minoritaria:
  - ◇ Precision: 8 casi di vittoria, 6 di pareggio e 3 di sconfitta;
  - ◇ Recall: 1 caso di vittoria, 8 di pareggio e 8 di sconfitta;
  - ◇ F1-Score: 7 casi di vittoria, 6 di pareggio e 4 di sconfitta.
- SMOTE con valutazione del dataset di training:
  - ◇ Precision: 0 casi di vittoria, 6 di pareggio e 11 di sconfitta;

- ◇ Recall: 3 casi di vittoria, 10 di pareggio e 4 di sconfitta;
- ◇ F1-Score: 1 caso di vittoria, 6 di pareggio e 10 di sconfitta.

Gli Alberi di Decisione evidenziano i primi limiti dell'approccio proposto.
Anche se si ha un numero significativo di vittorie, essi sono accompagnati da un quantitativo elevato di pareggi e di sconfitte.
Il decadimento delle prestazioni è visibile attraverso un'analisi dei casi di vittoria: ad eccezione dei dataset kr-vs-k-zero_vs_eight (39% di miglioramento) e kr-vs-k-one_vs_fifteen (17% di progresso), l'incremento può esser ritenuto poco significativo per dichiarare l'applicazione dello SMOTE con valutazione e rimozione della classe minoritaria predominante sullo SMOTE.
Anche in questo caso, lo SMOTE con valutazione e rimozione del dataset di training non evidenzia i risultati sperati.

Tabella 6.24: Risultati GaussianNB

	SMOTE			**Valutazione classe minoritaria**			**Valutazione dataset di training**		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
kr-vs-k-zero-one_vs_draw	54%	**78%**	64%	**61%**	66%	63%	60%	77%	**67%**
new-thyroid1	92%	**97%**	94%	**94%**	**97%**	**95%**	**94%**	**97%**	**95%**
vehicle3	41%	**69%**	**51%**	**43%**	61%	50%	41%	60%	49%
kr-vs-k-zero_vs_eight	13%	**59%**	21%	**92%**	44%	**60%**	17%	44%	25%
kr-vs-k-zero_vs_fifteen	76%	**81%**	**78%**	**100%**	63%	77%	**100%**	63%	77%
vehicle0	**40%**	92%	**56%**	39%	84%	53%	39%	**99%**	**56%**
ecoli1	**56%**	**90%**	69%	**56%**	**90%**	**70%**	50%	**90%**	64%
kr-vs-k-one_vs_fifteen	57%	**85%**	69%	**87%**	**85%**	**86%**	56%	**85%**	68%
kr-vs-k-three_vs_eleven	**100%**	85%	**92%**	69%	89%	78%	54%	**94%**	69%
ecoli3	**40%**	91%	**56%**	**40%**	**94%**	**56%**	35%	**94%**	51%
haberman	50%	37%	43%	**60%**	26%	36%	50%	**43%**	**46%**
glass6	**85%**	**79%**	**82%**	**85%**	**79%**	**82%**	81%	76%	78%
shuttle-2_vs_5	**11%**	**96%**	**20%**	**11%**	**96%**	**20%**	9%	**96%**	16%
car-vgood	20%	**100%**	33%	**21%**	**100%**	**35%**	20%	**100%**	33%
yeast3	**17%**	**97%**	**28%**	**17%**	**97%**	**28%**	13%	**97%**	23%
shuttle-6_vs_2-3	**91%**	**100%**	**95%**	**91%**	**100%**	**95%**	**91%**	**100%**	**95%**
car-good	11%	**59%**	**19%**	**22%**	17%	**19%**	8%	45%	14%
Average	50%	**82%**	57%	**58%**	75%	**59%**	48%	80%	54%

L'algoritmo *GaussianNB* presenta le seguenti caratteristiche:

- SMOTE:
  - ◇ Precision: 2 casi di vittoria, 6 di pareggio e 9 di sconfitta;
  - ◇ Recall: 5 casi di vittoria, 8 di pareggio e 4 di sconfitta;
  - ◇ F1-Score: 3 casi di vittoria, 7 di pareggio e 7 di sconfitta.

- SMOTE con valutazione della classe minoritaria:
  - ◇ Precision: 7 casi di vittoria, 8 di pareggio e 2 di sconfitta;
  - ◇ Recall: 0 casi di vittoria, 9 di pareggio e 8 di sconfitta;
  - ◇ F1-Score: 4 casi di vittoria, 7 di pareggio e 6 di sconfitta.
- SMOTE con valutazione del dataset di training:
  - ◇ Precision: 0 casi di vittoria, 3 di pareggio e 14 di sconfitta;
  - ◇ Recall: 3 casi di vittoria, 8 di pareggio e 6 di sconfitta;
  - ◇ F1-Score: 2 casi di vittoria, 3 di pareggio e 12 di sconfitta.

L'algoritmo GaussianNB rappresenta nettamente il worst case dell'applicazione dello SMOTE con valutazione e rimozione della classe minoritaria e SMOTE con valutazione e rimozione del dataset di training.
Nel primo caso, si evidenziano solamente 4 casi di vittoria, dando spazio ad un'ampia collezione di pareggi (7) e sconfitte (6).
Anche nel secondo, non si raggiungono i risultati sperati, con solamente 2 casi di vittoria, 3 di pareggio e ben 12 di sconfitta.

6.5 Commenti e conclusioni

Come evidenziato dai documenti presenti in letteratura, si è dimostrato che il Clustering aiuta la semplice classificazione.
Questa considerazione può esser ritenuta vera sia nel caso d'utilizzo della singola entry come dato di test sia nel caso in cui viene utilizzato per il test l'intero cluster.
Tuttavia, anche se si nota un incremento delle prestazioni, non si può esser soddisfatti del risultato finale.
Tale insoddisfazione è dettata da un valore di F1-Score troppo basso, che in un caso reale produrrebbe un grande numero di classificazioni errate.
Per tal motivo, si è optato per delle strategie alternative, che vedono come sistema alla base lo SMOTE.
A seguito di un'analisi della totalità dei casi di test, si conclude che la strategia SMOTE con valutazione e rimozione della classe minoritaria è la più efficace, in quanto risulta vincitrice in 43 dei 102 casi di test (42%).
Inoltre, si mettono in risalto 43 casi di pareggio (42%) e 16 di sconfitta (16%).
La situazione di pareggio non dovrà esser letta negativamente: è da considerarsi positiva una condizione in cui non vi è un degrado delle prestazioni, anche se non si ha l'incremento desiderato.
Si vuole mettere in risalto un aspetto: 43 rappresenta il numero di vittorie rispetto le altre due strategie.
Se si considera il solo SMOTE, il cui superamento delle prestazioni rappresenta l'obbiettivo principale, lo SMOTE con valutazione e rimozione della classe minoritaria può ritenersi vincente in ben 63 casi (62%), a cui vengono aggiunti 29 casi di pareggio (28%)

e 10 di sconfitta (10%).
In tutt'altro modo deve esser letto l'approccio SMOTE con valutazione e rimozione del dataset di training.
Questa strategia può esser considerata un fallimento, in quanto viene battuta o eguagliata per quasi la totalità dei casi dalla rivale SMOTE con valutazione e rimozione della classe minoritaria.
Il vantaggio apparente dell'eliminazione di dati che comportano un'errata classificazione durante la costruzione del modello sembrerebbe tradursi in un'eliminazione dal dataset di trainig di dati che in realtà risultano fondamentali per l'apprendimento.

È di fondamentale importanza capire la ragione intrinseca che comporta delle performance migliori nel caso delle Reti Neurali e delle SVM piuttosto che nel caso del classificatore Bayesiano o degli Alberi di Decisione.
Come spiegato nei paragrafi introduttivi relativi agli algoritmi di classificazione (Sezione 2.2), modelli come SVM e NN sono supportati alle spalle da modelli matematici complessi.
In relazione a quest'ultimo punto, nella Sezione 3.2.7 è stata illustrata una possibile analogia tra le NN e le SVM.
Si ricorda che lo SMOTE fondamentalmente sintetizza dei dati della classe minoritaria partendo da quelli realmente classificati come appartenenti a tale classe: immaginando di essere in possesso di due entry $X = (1, 2)$ e $Y = (3, 3)$, potenzialmente la fase di sintetizzazione darà vita al dato $S = (2, 2.5)$.
Si può ipotizzare che lo SMOTE (che di fatto consiste nell'aggiunta di dati numerici che potenziano la rilevanza della classe meno frequente) favorisca algoritmi come MLP e SVM in quanto essi lavorano bene con dati di tipo numerico.
In sostanza, durante la fase della costruzione di una Neural Network, i pesi che vanno a determinare le connessioni della rete tengono fortemente in considerazione i nuovi dati, perfezionati attraverso lo SMOTE con valutazione e rimozione della classe minoritaria.
Un ragionamento analogo può esser fatto per le SVM, dove gli iperpiani di separazione vengono posizionati prendendo in considerazione i nuovi punti nello spazio.
Anche nelle KNN, cambiare il numero di entry appartenente alla classe minoritaria implica incidere fortemente sul numero di vicini che potenzialmente sentenziano una decisione.
Discorso diverso deve esser fatto per gli Alberi di Decisione o il classificatore Bayesiano, che lavorano molto bene su dati strutturati (e.g. dati all'interno di una tabella).
La perdita del dato strutturato sembrerebbe incidere pesantemente sulle performance del classificatore, non permettendo un miglioramento delle tecniche di rimozione.

6.6 Risultati L^3 Cost Sensitive

L'obbiettivo di L^3 *Cost Sensitive* sarà dimostrare che una strategia basata sul concetto di costo miglirerà la classificazione base di L^3.
I risultati ottenuti sono riassunti nella Tabella 6.25.

Tabella 6.25: Risultati L^3 *Cost Sensitive*

	L3			***L^3 Cost Sensitive***		
	P	**R**	**F**	**P**	**R**	**F**
kr-vs-k-one_vs_draw	**92%**	75%	83%	**92%**	**86%**	**89%**
new-thyroid1	**96%**	74%	**83%**	76%	**87%**	81%
vehicle3	59%	32%	42%	**60%**	**45%**	**51%**
kr-vs-k-zero vs eight	**100%**	85%	92%	**100%**	**100%**	**100%**
kr-vs-k-zero vs fifteen	**100%**	92%	96%	**100%**	**100%**	**100%**
vehicle0	85%	**83%**	**84%**	**94%**	76%	**84%**
ecoli1	**75%**	53%	62%	59%	**76%**	**67%**
kr-vs-k-one vs fifteen	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
kr-vs-k-three vs eleven	**100%**	97%	98%	**100%**	**100%**	**100%**
ecoli3	**63%**	34%	44%	35%	**67%**	**46%**
haberman	**40%**	7%	12%	34%	**55%**	**42%**
glass6	**95%**	72%	82%	80%	**92%**	**86%**
page-blocks0	**92%**	4%	9%	37%	**88%**	**52%**
shuttle-2 vs 5	**100%**	81%	**89%**	81%	**100%**	**89%**
car-vgood	**97%**	66%	78%	80%	**100%**	**88%**
yeast3	**84%**	45%	59%	55%	**77%**	**64%**
car-good	**88%**	58%	**70%**	39%	**100%**	56%
Average	**87%**	61%	69%	72%	**86%**	**76%**

Si può ritenere raggiunto l'obbiettivo del superamento delle prestazioni di L^3 da parte di L^3 *Cost Sensitive*, in termini di F1-Score della classe minoritaria.
Si mettono in risalto i seguenti aspetti:

- In termini di Precisione, l'algoritmo L^3 presenta sicuramente dei risultati equiparabili o migliori.
  Tuttavia, come già detto, non bisogna soffermarsi alla valutazione della Precisione.
  Si prenda in considerazione la matrice di confusione riportata nella Tabella 6.26, tratta dalla validazione del dataset page-blocks0.

Tabella 6.26: Matrice di confusione page-blocks0

	Positivo	**Negativo**
Positivo	24	514
Negativo	4	4908

  Si registra il 92% di Precisione nella classificazione della classe minoritaria.
  Questo risultato non è da considerarsi vantaggioso, in quanto uno score così alto nasce dall'incapacità del classificatore di predire delle entry come appartenenti alla

classe minoritaria.
Si può infatti notare, analizzando le entry classificate come negative, che la tendenza del classificatore sarà etichettare tutti i dati come negativi (overfitting), alzando notevolmente il valore della Precisione ma abbassando lo score del Richiamo (solamente 4%).
Questo ci fa capire quanto sia importante valutare un risultato in termini di F1-Score.

- In termini di Richiamo, L^3 *Cost Sensitive* registra dei risultati notevoli.
  In questo caso, al contrario di L^3, Richiamo alto non implica Precisione bassa, ma si riescono a mantenere per entrambe le misure standard qualitativi abbastanza elevati.
  L'aumento del Richiamo a discapito della Precisione può considerarsi connesso con l'uso della matrice di costo.
  Per spiegare questo fenomeno, si pensi alla loro definizione: la Precisione è influenzata dal numero di *Falsi Positivi* (FP), viceversa al Richiamo è connesso il numero di *Falsi Negativi* (FN).
  Associare un peso alle regole sposta un elemento che prima poteva essere pensato come appartenente alla classe maggioritaria verso quella minoritaria, introducendo una correzione nella predizione di quel dato.

In ultimo, L^3 *Cost Sensitive* vince nel 71% dei casi (12 dataset su un totale di 17 di Benchmark) in termini di F1-Score.
Si evidenziano 3 situazioni di pareggio (18%) e 2 di sconfitta (11%).
Alla luce delle considerazioni appena effettuate, tali risultati possono considerarsi notevoli, in quanto viene migliorata (in media) l'F1-Score del 7%.
Inoltre, possono essere prese in considerazione alcune casistiche, come per i dataset haberman, page-block0 e car-vgood, dove si ha un aumento in termini di punti percentuali dell'F1-Score di più del 10%.

7 Conclusioni

Nell'ambito del Machine Learning, la classificazione dei dati ricopre un ruolo fondamentale.
La predizione è resa tanto più difficle quanto più complessa è la struttura del dataset, dove la complessità può esser misurata sulla base del numero e della tipologia di attributi e del numero di entry.
In questo senso, il numero di dati appartenenti ad ogni classe gioca un ruolo fondamentale, in quanto potrebbe indurre il classificatore a predire un dato incognito come appartenente alla classe per la quale sono presenti un maggior numero di elementi, introducendo così la problematica dell'overfitting.
I dataset in cui il numero di entry appartenenti ad una classe è di molto maggiore rispetto al numero di elementi delle altre classi vengono definiti come *dataset sbilanciati*.
Questa tesi, nell'ottica di una buona predizione in termini di Precision, Recall e F1-Score della classe minoritaria, si è proposta due obbiettivi chiave:

1. Applicazione di metodologie di classificazione ad un caso di studio reale.
   In particolare, si è analizzato un contesto di studio insurance, dove il tentativo è stato quello di prevedere se un cliente della compagnia assicurativa è esposto o meno a rischio di incidente in base al proprio stile di guida;

2. Ricerca di soluzioni alternative rispetto a quelle esistenti allo stato dell'arte, cercando di superarne i limiti riscontrati.
   In tal senso, sono state individuate delle soluzioni differenti in base ad un approccio di tipo supervisionato e non.

In un primo step di ricerca, una compagnia assicurativa ha messo a disposizione la propria banca dati al fine di capire se fosse possibile o meno classificare i propri utenti come soggetti a rischio, a seguito di un'analisi basata sul loro profilo e stile di guida.
L'esame di questo dataset può esser considerata di notevole difficoltà, in quanto il numero di entry della classe maggioritaria è nettamente maggiore rispetto al numero di dati della classe minoritaria (IR tra 57 e 256, a seconda dell'attributo di classe predetto).
Dopo una prima fase di indagine, atta a stabilire il miglior algoritmo e la migliore strategia di classificazione, l'approccio di tipo Undersampling è risultato il più efficace tra quelli proposti.
Questo risultato può ritenersi atteso a seguito di un'analisi del dataset: una grande predominanza della classe maggioritaria ha reso impossibile l'applicazione dell'Oversampling, in quanto non è pensabile replicare o sintetizzare un elevato numero di entry.
I risultati dell'Undersampling sono stati notevoli, in quanto si è registrato un incremento delle performance in termini di F1-Score (sulla base dell'attributo predetto) tra il 44% e il 78%.
In riferimento al punto 2, le proprie tecniche sono state validate su dei dataset sbilanciati di Benchmark disponibili sul repository UCI.
Si è dimostrato che, le soluzioni basate su Clustering presenti in letteratura, migliora-

no notevolmente l'algoritmo base utilizzato per la classificazione ma non comportano il raggiungimento di un buon risultato.
Un approccio di tipo SMOTE Oversampling offre nella sua implementazione base dei risultati migliori.
Lo SMOTE sintetizza dei dati appartenenti alla classe minoritaria partendo da quelli già a disposizione all'interno del dataset, riducendo così lo sbilanciamento tra classi.
Poiché i dati vengono sintetizzati in punti che giacciono sulla retta di congiunzione creata tra elementi differenti, non si avrà nessuna garanzia che il dato appartenga realmente alla classe minoritaria.
La tecnica dello SMOTE con valutazione e rimozione degli elementi della classe minoritaria offre l'opportunità di rimuovere i dati sintetizzati come appartenenti alla classe minoritaria ma in realtà potenzialmente inerenti ad un'altra classe.
Questo ha permesso, in fase di apprendimento, la possibilità di costruire un modello sulla base di un dataset di training più solido.
Da un'analisi dei risultati, si evince che su 102 casi di test, le performance vengono migliorate rispetto al semplice SMOTE in 63 casi (62%).
Le due strategie invece possono essere ritenute comparabili in 29 casi (28%).
Infine, lo SMOTE offre delle prestazioni migliori rispetto alla variazione con valutazione e rimozione degli elementi della classe minoritaria in 10 casi di test (10%).
In un'ottica di miglioramento, la modifica all'algoritmo associativo L^3 ha ricoperto un altro aspetto fondamentale di questa tesi.
Partendo dalla considerazione per cui si riscontra una difficoltà nella classificazione degli elementi della classe minoritaria, un approccio di tipo *Cost-Sensitive* consente di associare un peso alle regole della classe minoritaria così da esser prese maggiormente in considerazione.
Sarà proprio il peso (nel gergo *rischio*) che influenzerà la scelta del classificatore.
L'utilizzo della matrice di costo comporta un incremento medio del 7% in termini di F1-Score, accrescendolo 12 volte su 17 (71%).
Il punto di forza di quest'approccio risiede nell'aumento del valore di Richiamo, lasciando tuttavia inalterato (o quasi) il valore della Precisione.

Riferimenti bibliografici

[1] Marie-Josée Proulx, Eveline Bernier e Yvan Bédard. "Environmental Health Systematic Review". In: *ResearchGate* (2019).

[2] URL: http://dbdmg.polito.it/wordpress/wp-content/uploads/2018/10/9-DMClustering.pdf.

[3] URL: http://dbdmg.polito.it/wordpress/wp-content/uploads/2018/10/7-DMassrules.pdf.

[4] O. Agrawal, T. Imielinski e A. Swami. "Mining Association Rules Between Sets of Items in Large Databases". In: *SIGMOD Conference: 207-216* (1993).

[5] J. Dongre, G. Lai Prajapati e S. V. Tokekar. "The role of Apriori algorithm for finding the association rules in Data mining". In: *IEEE* (2014).

[6] Y. Liu e Y. Guan. "FP-Growth Algorithm for Application in Research of Market Basket Analysis". In: *IEEE* (2008).

[7] *Introduction to Data Mining*. Michael, S. e Vipin, K., 2005.

[8] URL: https://areeweb.polito.it/didattica/gcia/.

[9] URL: https://en.wikipedia.org/wiki/Support_vector_machine.

[10] URL: https://it.wikipedia.org/wiki/K-nearest_neighbors.

[11] URL: https://scikit-learn.org.

[12] URL: https://archive.ics.uci.edu/ml/datasets.html.

[13] URL: http://rikunert.com/SMOTE_explained.

[14] URL: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html.

[15] URL: https://www.cs.waikato.ac.nz/ml/weka/.

[16] Baralis, E. and Chiusano, S. and Garza, P. "A Lazy Approach to Associative Classification". In: *IEEE* (2007).

[17] G. Dong et al. "Classification by Aggregating Emerging Patters". In: *Second Int'l Conf. Discovery Science* (1999).

[18] W. Li, J. Han e J. Pei. "CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules". In: *IEEE* (2001).

[19] B. Liu, W. Hsu e Y. Ma. "Integrating Classification and Association Rule Mining". In: *Fourth Int'l Conf. Knowledge Discovery and Data Mining* (1998).

[20] K. Wang, S. Zhou e Y. He. "Growing Decision Trees on Support-Less Association Rules". In: *Sixth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining* (2000).

[21] URL: www.eclipse.org/windowbuilder.

[22] URL: https://docs.oracle.com/javase/8/docs/api/index.html?javax/swing/package-summary.html.

[23] URL: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html.