POLITECNICO DI TORINO
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Tecnica di compressione finalizzata alla classificazione di dati

Relatore
prof. Paolo Garza

Candidato
Gaetano Valerio D'Aiera

Ottobre 2017

Ringraziamenti

Ringrazio anzitutto il mio relatore, il Chiar.mo Professore Paolo Garza, per la sua gentile disponibilità nel propormi il progetto e seguirmi durante l'implementazione dell'algoritmo proposto in questa Tesi di Laurea. Un particolare ringraziamento va a tutti coloro che mi hanno sostenuto durante tutto il mio percorso accademico.

Indice

Capitolo 1
Introduzione

Questo capitolo sarà dedicato all'introduzione del contenuto della tesi. Inoltre, verrà spiegato anche l'obiettivo e la struttura di questa tesi.

1.1 Background: problemi legati alla classificazione di grandi moli di dati

Negli ultimi anni vengono generati quantità di dati sempre più grandi da parte dei social network, dei dispositivi IoT e da tutte le altre fonti attualmente disponibili. Le aziende utilizzano questi grandi moli di dati prodotte per fornire dei servizi; infatti, applicando delle tecniche di analisi riescono ad estrarre informazioni utili sia per rendersi conto del proprio andamento attuale che per predire quello futuro in modo da prendere delle decisioni opportune ed evitare ad esempio rischi come quelli legati agli investimenti. Esistono diverse tecniche che permettono l'applicazione di questo tipo di analisi e sono conosciute sotto il nome di tecniche di Data Mining, temine che verrà introdotto e spiegato nel Capitolo 2 al Paragrafo 2.1.
Data l'enorme mole di dati prodotti, si vengono a creare delle problematiche durante il processo di analisi. In questo paragrafo verranno presi in considerazione i problemi legati alla fase di classificazione del processo di analisi, tematica che verrà ampiamente trattata nel Capitolo 2. La classificazione di insiemi di dati costituiti da un'elevata cardinalità portano a problematiche di notevole importanza. Considerando le attuali tecniche di classificazione, la maggior parte di esse ha la necessità di utilizzare la memoria principale dell'unità di elaborazione per consentire l'apprendimento del classificatore, ma dato il numero elevato di dati, occorre utilizzare delle memorie di capacità sempre maggiori e questo non è sempre possibile, dato anche l'elevato costo. Quindi, maggiore è la quantità di dati, maggiore sarà la quantità di memoria richiesta per poterne effettuare la classificazione. Questo è solo uno dei problemi che può presentare la classificazione di grandi moli di dati. Infatti, un'altra problematica è associata all'elevata quantità di tempo impiegato dall'unità di elaborazione per poter generare i risultati della classificazione. Quindi, se fosse possibile effettuare la classificazione con la memoria disponibile sull'unità elaborativa, bisognerebbe stimare l'effettivo tempo impiegato nella classificazione. Inoltre, per procedere alla classificazione, occorre dapprima la costruzione di un modello in grado di classificare successivamente i dati, questo introduce ulteriori problematiche oltre a quelle presentate in precedenza. Oltre a tenere conto del tempo impiegato dal processo di classificazione, infatti, bisogna considerare anche il tempo necessario per la costruzione di questo modello.
Un'altra problematica legata alla classificazione di grandi moli di dati è l'eccessiva complessità del modello di classificazione costruito utilizzando per intero la collezione di

dati che può portare al fenomeno del cosiddetto overfitting, che sarà analizzato in maniera opportuna nel Paragrafo 2.3.1.2 del Capitolo 2. In aggiunta, la costruzione di un modello di classificazione tramite l'utilizzo di una grande moli di dati potrebbe non essere possibile a causa dei limiti di memoria di un'unità elaborativa.
Dopo aver trattato tutti i problemi legati alla classificazione di collezioni di dati di grandi dimensioni è possibile discutere riguardo le contromisure che si possono adottare per far fronte a queste problematiche. Nella letteratura è possibile identificarne due diverse tipologie che verranno presentate di seguito.
Una prima soluzione sarebbe quella legata all'uso di architetture distribuite per il salvataggio e la successiva analisi di queste grandi moli di dati, visto che le tradizionali tecniche di classificazione non sono efficienti o talvolta addirittura inutilizzabili. Questo tipo di architetture scalano in accordo alla dimensione di queste collezioni di dati. Per poter procedere alla classificazione occorre utilizzare degli algoritmi di classificazione parallelizzabili visto che i dati, in architetture di questo tipo, vengono divisi in porzioni su varie unità elaborative. Ognuna di queste unità di elaborazione analizza la porzione di dati ad essa associata e fornisce un risultato parziale che viene successivamente combinato con i risultati generati da tutte le altre per arrivare ad un risultato finale di classificazione. Se si volesse utilizzare un approccio di questo tipo risulta necessario l'uso di framework noti che consentono l'applicazione di tecniche di classificazione per Big Data, che saranno presentate nel Capitolo 2 al Paragrafo 2.4.4.
Una soluzione differente, invece, è quella di ridurre in qualche modo la collezione di dati di grandi dimensioni così da poter utilizzare le tradizionali tecniche di classificazione, poiché queste, in aggiunta, consentono di ottenere elevati valori di accuratezza nella classificazione. L'idea è quella di utilizzare delle tecniche di riduzione in modo tale da riuscire a rappresentare l'enorme quantità di dati in una versione compressa. Occorre tenere presente che la compressione deve portare alla costruzione di un modello in grado di produrre una qualità di classificazione simile a quella ottenuta utilizzando l'insieme di dati iniziali di grandi dimensioni.
Tra le tecniche di riduzione presenti si trovano quelle che consentono la riduzione del numero di attributi di una collezione di dati oppure quelle che permettono di rappresentare la collezione di dati iniziale in una forma ridotta, dal punto di vista della cardinalità, applicando una qualche funzione.

1.2 Obiettivo della tesi

Date le problematiche discusse nel Paragrafo 1.1, in questa tesi è proposta una soluzione che consente la riduzione, in termini di cardinalità, di una collezione di dati di grandi dimensioni. L'obiettivo di questa tesi, quindi, è la realizzazione di un algoritmo in grado di generare una versione compressa di una collezione di dati iniziale in modo da non incorrere in un'eccessiva perdita di accuratezza della classificazione da parte del modello generato da questa nuova versione compressa della collezione di dati. Quindi, in

definitiva, l'algoritmo proposto è utile per la compressione di dati etichettati finalizzato alla classificazione di grandi moli di dati.

1.3 Struttura della tesi

La tesi è organizzata in cinque capitoli, di cui questo capitolo di introduzione ne costituisce il Capitolo 1. Nel capitolo successivo, il Capitolo 2, sarà introdotto il concetto di Data Mining con particolare attenzione rivolta alla parte riguardante la classificazione. Inoltre, verranno presentate ed analizzate tutte le tecniche di classificazione attualmente disponibili. Nel Capitolo 3 verrà presentato il lavoro svolto durante il percorso di tesi ed analizzato il funzionamento dell'algoritmo implementato. Nel Capitolo 4 verranno presentati tutti i risultati ottenuti dagli esperimenti effettuati utilizzando l'algoritmo proposto in questa tesi. Infine, nel Capitolo 5, sarà dato un commento finale sul percorso di tesi svolto e sulla qualità dell'algoritmo implementato. Inoltre, verranno proposti possibili sviluppi futuri.

Capitolo 2
Stato dell'arte

In questo capitolo, dopo un'iniziale introduzione al concetto di Data Mining, con particolare attenzione rivolta alla classificazione, verranno presentate le principali tecniche di classificazione attualmente disponibili sia per dati di dimensioni non big sia per i Big Data.

2.1 Data Mining

Con il termine Data Mining si identificano un insieme di tecniche di analisi che permettono un processo automatico di estrazione di informazioni utili, chiamati **pattern**, da collezioni di dati (o **dataset**). Queste tecniche di estrazione automatica si basano sull'utilizzo di diversi algoritmi. [1]

Figura 2.1 Processo di estrazione di informazioni utili dai dati

Per poter arrivare all'estrazione di pattern è necessario seguire un processo diviso in varie fasi, chiamato **knowledge discovery process**, che inizialmente parte dalla selezione dei dati da sottoporre ad analisi dalle diverse fonti disponibili, come ad esempio dei dati presi dalle basi di dati. Successivamente si passa in una fase di fondamentale importanza per l'accuratezza finale dei pattern estratti, la fase di **preprocessing**. Infatti, questa fase permette di migliorare la qualità dei dati ad esempio risolvendo delle inconsistenze che questi possono presentare, rimuovendo eventuali dati anomali (**outliers**) non utili all'analisi, così chiamati poiché si differenziano dal resto dei dati all'interno del dataset, e risolvendo eventuali conflitti presenti tra i dati dopo l'integrazione delle diverse fonti. Questa fase di preprocessing è seguita da una di trasformazione che consente di preparare i dati per alla fase successiva di Data Mining, grazie alla quale, come precedentemente detto, è possibile estrarre i pattern dai dati applicando degli specifici algoritmi. Infine, dopo una fase di interpretazione e valutazione dei pattern estratti, si passa alla fase finale chiamata **knowledge**, in cui i dati finali delle analisi vengono presentati.
Attualmente, i contesti di utilizzo delle tecniche di Data Mining sono diversi: alcuni esempi sono l'analisi di dati estratti dai social network, la comprensione del

comportamento di utenti in siti Web, come ad esempio gli e-commerce, l'analisi dei testi (**text mining**), che permette la conoscenza di informazioni utili estratte dai documenti, l'analisi di dati clinici, la fornitura di servizi alle persone attraverso l'uso della loro posizione e vari altri contesti.

2.1.1 Tipologie di dati e attributi delle collezioni di dati

Esistono diverse tipologie di collezione di dati tra cui comunemente si trovano quelle costituite da dati caratterizzati da diversi attributi. In una collezione di dati di questo tipo, ogni dato presenta i valori per ogni attributo. Ogni attributo ne rappresenta una determinata caratteristica e ne esistono diverse tipologie tra cui troviamo quelli:

- Nominali: presentano un valore che corrisponde ad una caratteristica (es. i colori)
- Ordinali: i valori di un attributo di questo tipo sono ordinati tra loro
- Numerici (o contigui): presentano valori numerici interi reali
- Binari: gli attributi di questo tipo sono costituiti da due soli valori

Nelle collezioni di dati, può capitare che non tutti i dati presentano tutti i valori per tutti gli attributi che li caratterizzano. Infatti, spesso nei dataset è possibile identificare dei dati che presentano i cosiddetti **missing value**, valori mancanti per un determinato attributo. Una collezione di dati caratterizzata da questi valori mancanti può influenzare la qualità delle analisi che verranno effettuate su di essa; infatti, durante il processo di analisi o viene eliminato l'intero dato che li contiene oppure il dato viene mantenuto, ma i valori mancanti vengono sostituiti da altri valori.

2.2 Classificazione

Nel Data Mining, un processo di classificazione consiste nell'assegnazione di etichette di classe ai dati non ancora etichettati. Dato un insieme di classi, individuate per mezzo di **etichette di classe**, e una collezione di dati precedentemente classificati, l'obiettivo è quello di generare un **modello di classificazione** che consenta di assegnare le corrette etichette di classe ai dati non ancora etichettati. Un modello di classificazione viene generato utilizzando uno specifico **algoritmo di classificazione** a cui in input vanno dei **dati di training** (o **training set**) che consistono in una collezione di dati, chiamati **istanze**, costituiti da un insieme di valori per un insieme finito attributi e da un valore per l'attributo di classe. Dall'analisi dei dati di training viene generato il modello di classificazione e questo conclude la prima fase del processo di classificazione. Successivamente, utilizzando questo modello di classificazione è possibile assegnare ai dati non ancora etichettati, denominati **dati di test** (o **test set**), le rispettive etichette di classe. Tutto il procedimento spiegato è mostrato nello schema della Figura 2.2:

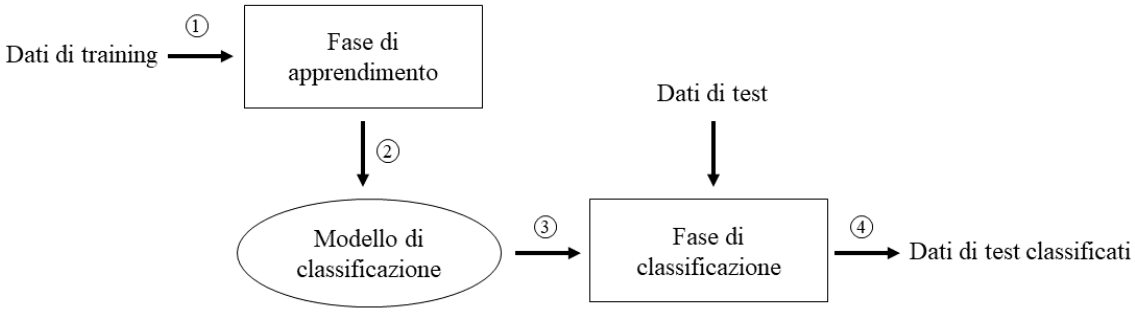


Figura 2.2 Schema classificazione

Dalla Figura 2.2 si evince che dopo una fase di apprendimento utilizzando come input le istanze di training viene generato un modello di classificazione. Questo prende in ingresso le istanze di test e viene avviata la fase di classificazione. Alla fine di tutto il processo di analisi tutte le istanze di test vengono etichettate tramite un'etichetta di classe.
La classificazione è divisa in due tipologie: **supervised** e **unsupervised** (o **clustering**). Nella classificazione di tipo supervised l'insieme delle possibili etichette di classi è conosciuta a priori, quindi, nel processo di classificazione, alle istanze di test viene assegnata un valore di etichetta di classe tra quelle indicate a priori. Invece, nella classificazione di tipo unsupervised non ci sono etichette di classe note a priori, ma solo dopo aver effettuato la classificazione, dividendo i dati in gruppi secondo attributi simili, è possibile dare un nome ai diversi gruppi generati e creare così le etichette di classe. Quest'ultima è meglio nota sotto il nome di **clustering**.

2.2.1 Preparazione dei dati

Per poter generare un modello di classificazione dotato di una buona qualità nel classificare le istanze di test, occorre effettuare una fase di preprocessing della collezione di dati in input. Infatti, i dati che vengono solitamente analizzati provengono da diverse fonti e possono quindi presentare ridondanze, dati anomali o mancanti, rumori o conflitti. Per limitare l'influenza di tutte queste problematiche sul risultato finale della classificazione occorre utilizzare delle tecniche di preprocessing.

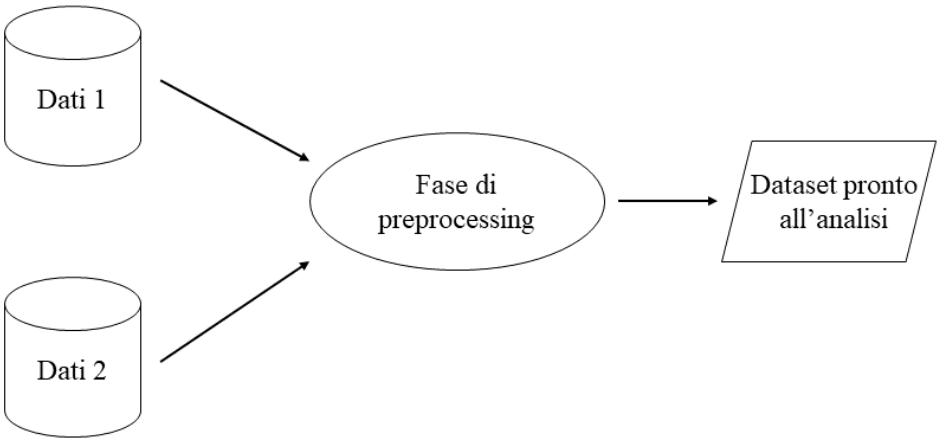


Figura 2.3 Schema preprocessing

Nella Figura 2.3 è mostrato uno schema in cui i dati vengono prelevati da due diverse fonti indicate come "Dati 1" e "Dati 2" e, dopo una fase di preprocessing, vengono generata una collezione di dati pronti per la fase successiva del processo di analisi.

2.2.2 Valutazione della qualità di un classificatore

L'obiettivo finale della classificazione è quello di etichettare le istanze di test grazie all'utilizzo del modello di classificazione creato dai dati di training. Quindi, occorre valutare il comportamento di un classificatore utilizzando alcune misure di qualità. Solitamente, la qualità di un classificatore viene valutata in termini di:

- **Accuratezza**: indica la qualità della classificazione dei dati non etichettati; è definita come la percentuale di dati correttamente classificati rispetto al numero totale di dati sottoposti a classificazione;
- **Efficienza**: tempo di generazione del modello di classificazione utilizzando i dati di training e tempo di classificazione dei dati di test;
- **Scalabilità**: come si comporta un classificatore al variare della dimensione e del numero di attributi del training set;
- **Robustezza**: indica quanto un classificatore è in grado di essere robusto in presenza di collezioni di dati contenenti missing value o **rumori**, definiti come istanze di training etichettate in modo errato;
- **Interpretabilità**: livello di interpretabilità del modello di classificazione.

2.3 Descrizione e analisi delle tipologie degli algoritmi di classificazione

Nello stato dell'arte esistono diverse tecniche di classificazione di tipo supervised, tra le quali sono presenti: alberi di decisione, regole di classificazione, regole di associazione, reti neurali, naïve bayes e reti bayesiane, k-nearest neighbours (k-NN), support vector machine (SVM). Nei paragrafi successivi verranno descritte e analizzate queste tecniche e verranno anche valutate in accordo ai termini di qualità presentati nel Paragrafo 2.2.2.

2.3.1 Alberi di decisione

La tecnica di classificazione basata sull'albero di decisione prevede la costruzione di un albero che presenta tre principali elementi: i **nodi**, gli **archi** e le **foglie**. [2]
Un nodo è costituito da un **attributo di split**, attributo scelto tra quelli dei dati di training secondo una strategia di tipo greedy, miglior attributo locale in grado di produrre una distribuzione omogenea dei dati di training in accordo con le etichette di classe. Ogni **arco** consente la connessione tra un nodo padre e un nodo figlio e presenta uno specifico valore dell'attributo di split presente nel nodo padre. Ogni **foglia**, invece, rappresenta un'etichetta di classe.

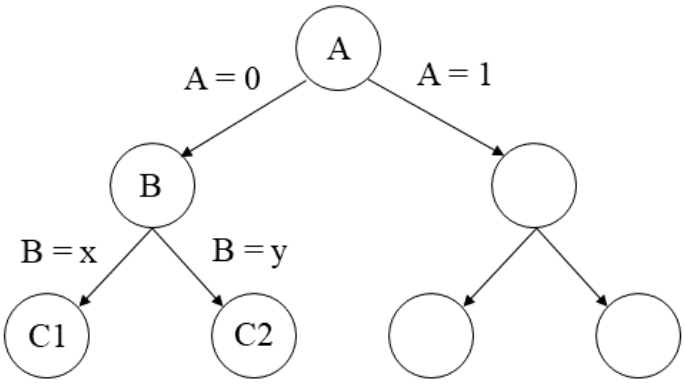


Figura 2.4 Albero di decisione

Nella Figura 2.4 è mostrato un esempio di albero di decisione in cui il nodo radice rappresenta l'attributo A dell'insieme di dati di training e i due archi uscenti dal questo nodo presentano rispettivamente uno tra i possibili valori che l'attributo A può assumere. A seconda del valore che un'istanza di test presenta per l'attributo A verrà percorso l'arco di sinistra o quello di destra dell'albero di decisione durante la fase di classificazione. Inoltre, percorrendo il ramo di sinistra si incontra il nodo figlio B del nodo radice che a sua volta genera due nodi foglia che costituiscono due valori possibili valori di etichette di classe: C1 e C2.

Ogni nodo dell'albero di decisione è una condizione da testare per ogni istanza dei dati di test e può presentare una struttura di due tipi: **multi-way split**, che consente di generare un arco per ogni valore distinto dell'attributo scelto per lo split o **binary split** che permette di dividere i valori dell'attributo di split soltanto in due sottoinsiemi. Nella Figura 2.5 sono mostrati le due possibili strutture di un nodo.

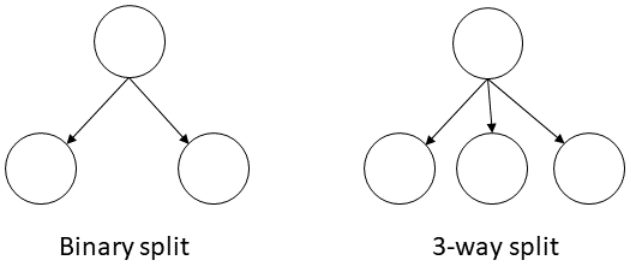


Figura 2.5 Strutture di un nodo

Anche se un attributo può assumere più di due valori, ma si sceglie di effettuare un split di tipo binario, i valori possono essere raggruppati in diversi modi per poter consentire lo split binario.
La costruzione del modello di classificazione ad albero differisce a seconda dell'algoritmo utilizzato e ne esistono diversi tra cui: algoritmo di Hunt, ID3, C4.5, C5.0, SLIQ, SPRINT, CART.

2.3.1.1 Costruzione di un albero di decisione

La costruzione di un albero di decisione avviene grazie all'utilizzo della collezione dei dati di training. Inizialmente, analizzando i dati di training, viene selezionato il migliore attributo di split per il nodo radice, utilizzando degli indici di qualità che saranno introdotti nel Paragrafo 2.3.1.4. In seguito, lo split viene eseguito sull'attributo scelto e i dati vengono divisi in un numero di sottoinsiemi pari al numero di archi creati dallo split. Per esempio nel caso di split binario, i dati di training saranno divisi soltanto in due sottoinsiemi. Occorre notare che i dati vengono divisi a seconda del valore che assumono per l'attributo su cui viene effettuato lo split. Infine, questo procedimento viene ripetuto in maniera sui vari sottoinsiemi dei dati di training che si vengono a creare dagli split, fino a quando non viene raggiunta una **condizione di fine** dell'algoritmo utilizzato; ad esempio fino a quando non ci sono più dati da separare. A questo punto la generazione dell'albero di decisione è completa.

2.3.1.2 Problematiche di costruzione di un modello: overfitting e underfitting

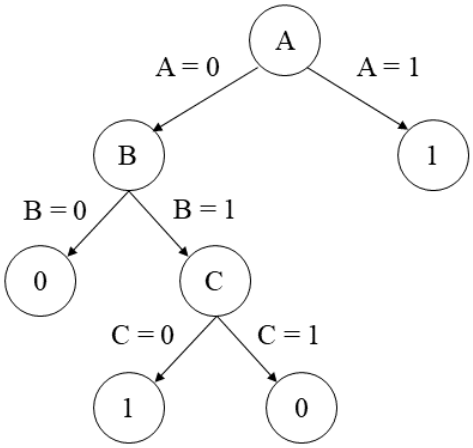
La costruzione di un modello di classificazione ad albero di decisione può portare a delle problematiche note. Una tra queste è il fenomeno dell'**overfitting** che si viene a creare quando il modello di classificazione è troppo specializzato per i dati di training. Questo porta alla creazione di un albero di decisione complesso che presenta un numero elevato di nodi. All'aumentare del numero di nodi il valore di accuratezza della classificazione aumenta se si usano come dati di test le stesse istanze di training. Contrariamente, utilizzando delle istanze diverse come dati di test, il valore di accuratezza diminuisce all'aumentare del numero di nodi. Il fenomeno di overfitting può essere in parte risolto utilizzando le seguenti tecniche di potatura dell'albero di decisione:

- **Pre-pruning**: in una tecnica di questo tipo si parte dall'idea di interrompere la costruzione dell'albero di decisione quando si raggiungono determinate condizioni. Normalmente un algoritmo di costruzione dell'albero di decisione si arresta quando viene raggiunta una condizione di fine, come ad esempio quando tutte le istanze nei vari sottoinsiemi appartengo alla stessa classe. In una tecnica di pre-pruning si cerca di scegliere una soglia in modo tale da interrompere la costruzione dell'albero quando il numero di istanze dei vari sottoinsiemi si trova al di sotto di tale soglia scelta;
- **Post-pruning**: considerando questa tipologia di soluzione, invece, l'albero di decisione viene interamente costruito e poi viene potato tenendo conto del miglioramento delle prestazioni dopo le eventuali potature dei nodi apportate.

Utilizzando in maniera eccessiva queste tecniche utili alla riduzione dell'overfitting si può incorrere ad un'altra problematica, l'**underfitting**. Infatti, si parla di underfitting quando il modello di classificazione è stato generato in maniera troppo semplicistica ed il processo di classificazione sia dei dati di training che del test set porta in entrambi i casi ad un errore di classificazione elevato, quindi ad una bassa accuratezza nella classificazione.

2.3.1.3 Funzionamento della classificazione basata sull'albero di decisione

Per effettuare una classificazione utilizzando un albero di decisione occorre innanzitutto una fase di creazione del modello utilizzando le istanze dei dati di training. Successivamente è possibile classificare le istanze di test grazie al modello di classificazione precedentemente creato. Un esempio di funzionamento della classificazione basata sull'albero di decisione è il seguente:
Dato il modello di classificazione ad albero di decisione di esempio e le due istanze di test mostrate in Figura 2.6:



A	B	C	classe
1	0	0	?
0	1	1	?

Figura 2.6 Esempio di classificazione basata sull'albero di decisione

Le due istanze di test mostrate in Figura 2.6 sono costituite da quattro attributi "A", "B", "C", "classe". Durante la fase di classificazione, la prima istanza di test, dato che per l'attributo A presenta un valore uguale a 1, viene direttamente classificata con l'etichetta di classe 1, perché percorre l'arco di destra uscente dal nodo radice. Quindi, assumerà un valore 1 per l'attributo "classe". Invece, considerando la seconda istanza da etichettare, percorre l'arco sinistro che collega il nodo radice A con il nodo che rappresenta l'attributo di split B. In questo nodo viene testato il valore che assume l'istanza per l'attributo B che in questo caso è pari a 1, quindi non si raggiunge ancora un nodo foglia ed è necessario un ulteriore test sull'attributo C. Dopo questo ultimo test, l'istanza viene classificata con l'etichetta di classe 0, poiché il suo valore per l'attributo C è 1 e viene attraversato l'arco destro per raggiungere il nodo foglia. Il suo attributo "classe", quindi, assumerà il valore 0.

2.3.1.4 Indici di qualità dell'attributo di split

La selezione dell'attributo di split è un'operazione critica, in quanto bisogna scegliere tra tutti gli attributi quelli che consentono di produrre una distribuzione omogenea dei dati di training secondo le etichette di classe. Per valutare la qualità dello split occorre una **misura di impurità**, definita come la frequenza delle etichette di classe in quel determinato split. Un split presenta un elevato livello di qualità, quindi distribuzione omogenea in accordo con le etichette di classe, quando più basso è il suo livello di impurità. Esistono diverse misure dell'impurità di uno split, ognuna di queste viene utilizzata in base dell'algoritmo di costruzione dell'albero di decisione scelto. Tra gli

indici che misurano l'impurità di uno split sono presenti: Gini index, entropia ed errore di classificazione.
Il **Gini index** misura il livello di non omogeneità della distribuzione dei dati di training secondo le etichette di classe ed è un numero compreso tra 0 e 1. Più basso è il valore di questo indice più la distribuzione dei dati è omogenea, quindi indica un basso livello di impurità e conseguentemente un'alta qualità dello split. L'indice di Gini per un dato nodo n è definito come:

$$GINI(n) = 1 - \sum_{j} [p(j|n)]^2$$

dove $p(j|n)$ è la frequenza della classe j al nodo n.
Il valore massimo indica che tutte le istanze sono equamente distribuite tra tutte le classi e comporta un alto livello di impurità, invece, il valore minimo corrisponde ad un basso livello di impurità poiché tutte le istanze appartengono ad un'unica classe.
Quando un nodo *x* su cui occorre effettuare uno split genera *k* archi, quindi *k* nodi figli, la qualità dello split è calcolata:

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI\ (i)$$

dove n_i è il numero di record al nodo figlio i e n è il numero di record del nodo *x* scelto.
Questo indice è utilizzato da algoritmi per la costruzione del modello basato sull'albero di decisione come CART, SLIQ e SPRINT.
L'**entropia**, come l'indice di Gini, misura il livello di impurità di un nodo, ma consente di raggiungere il valore più basso più velocemente se più nodi figli appartengono alla stessa classe. Ad un dato nodo *n*, l'entropia è definita come:

$$ENTROPIA(n) = -\sum_{j} p(j|t)\log p(j|n)$$

dove $p(j|t)$ è la frequenza della classe j al nodo n.
Come nell'indice di Gini, un valore elevato dell'entropia corrisponde ad un alto livello di impurità, invece, uno basso indica un basso livello di impurità.
Dato un nodo padre p, il cui split genera k partizioni, si sceglie lo split che massimizza il seguente valore di guadagno, chiamato **information gain**:

$$GAIN_{split} = ENTROPIA(p) - (\sum_{i=1}^{k} \frac{n_i}{n} ENTROPIA(i))$$

dove n_i è il numero di record nella partizione i.
Questo indice di guadagno viene utilizzato da algoritmi come ID3 e C4.5, ma presenta lo svantaggio di tendere a preferire la creazione di un numero elevato di partizioni piccole ma con un basso livello di impurità. Per superare questo svantaggio è stato definito un altro guadagno, il **gain ratio**, definito come:

$$GAIN_{ratio} = \frac{GAIN_{split}}{-\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}}$$

dove il denominatore consente di evitare la creazione di tante piccole partizioni.
L'**errore di classificazione** misura il numero di record erroneamente classificati ad un certo nodo n, ed è definito come:

$$ERRORE(n) = 1 - \max p(i|n)$$

dove $p(i|t)$ indica la frequenza della classe i al nodo n.
Secondo l'errore di classificazione uno split su un nodo è di qualità maggiore quando il numero di errori minore per quel nodo. Come le due misure di impurità descritte in precedenza, un elevato valore di questo indice indica un numero elevato di errori, quindi i record sono distribuiti in modo equo tra le classi ed il valore di impurità è alto. Un valore basso dell'errore di classificazione, invece, indica un basso livello di impurità.

2.3.1.5 Qualità della classificazione basata sugli alberi di decisione

Come discusso nel Paragrafo 2.2.2, la valutazione della qualità delle tecniche di classificazione viene effettuata in base a determinate caratteristiche come accuratezza, efficienza, scalabilità, interpretabilità e robustezza. La tecnica di classificazione basata sugli alberi di decisione presenta, come tutte le altre tecniche, dei vantaggi e degli svantaggi. Tra i vantaggi ci sono:

- Efficienza: rapida costruzione del modello di classificazione, poiché non esplora tutte le possibilità ma opera secondo un approccio greedy (ottimo locale per ogni nodo dell'albero). Inoltre presenta una veloce classificazione di istanze di test, dato che occorre prendere un'istanza e propagarla nel modello ad albero per effettuarne la classificazione.
- Interpretabilità: modello facile da interpretare se è di piccole dimensioni
- Accuratezza: quando il problema di classificazione è semplice, l'accuratezza è comparabile ad altri metodi di classificazione
- Scalabilità

Uno svantaggio, invece, è il basso livello di robustezza, visto che in presenza di missing value l'accuratezza potrebbe diminuire.

2.3.2 Classificazione basata su regole

La classificazione basata su regole si basa sulla costruzione e sull'utilizzo di regole costituite da una o più **condizioni** e da un'etichetta di classe, chiamata anche **conseguente della regola**. [2] Un esempio di una regola di classificazione è mostrato di seguito:

(Condizioni) -> etichetta di classe

Le condizioni di una regola corrispondono ad uno o più attributi scelti tra quelli presenti tra i dati di training associati ai corrispondenti valori. Su questi attributi si effettua il test e sono tutti posti tra loro in una condizione di AND logico. Se tutte le condizioni di una determinata regola sono soddisfatte, allora viene assegnata l'etichetta di classe di quella regola all'istanza su cui viene effettuato il test.
Un esempio di regole di associazione è il seguente, in cui sono mostrate due diverse regole R1 e R2:

R1: *attributo-1 = A AND attributo-2 = A -> etichetta di classe = 0*
R2: *attributo-1 = A AND attributo-2 = B -> etichetta di classe = 1*

Un esempio di applicazione di queste regole può essere effettuato considerando la seguente istanza di test:

attributo-1	attributo-2	etichetta di classe
A	A	?

Questa istanza soddisfa la regola di classificazione R1, quindi l'etichetta di classe assegnata sarà 0.

2.3.2.1 Proprietà e problematiche delle regole

Le regole possono presentare le due proprietà seguenti:

- Mutualmente esclusive: quando ogni istanza soddisfa una e una sola regola
- Esaustive: ogni istanza soddisfa almeno una regola

Solitamente le regole sono sia mutualmente esclusive che esaustive, ma nel caso in cui una di queste proprietà non fosse soddisfatta si verrebbero a creare delle problematiche per la classificazione dei dati di test. Infatti, se le regole non fossero mutualmente esclusive, più di una regola soddisferebbe una determinata istanza di test; in questo caso si presenta il problema di quale regola occorre scegliere. Esistono diverse strategie da poter adottare in questi casi, tra cui è possibile utilizzare una soluzione chiamata **ordered rule set**, che consente di ordinare le regole di classificazione in base ad una priorità. Quindi, quando un'istanza di test soddisfa due o più regole verrà considerata la regola a priorità più elevata e all'istanza verrà assegnata l'etichetta di classe di quella regola. Se, invece, le regole non fossero esaustive, un'istanza potrebbe non soddisfare alcuna regola presente nell'insieme delle regole create. Una soluzione a questo problema è l'utilizzo di un'etichetta di classe di default in modo che se nessuna regola fosse soddisfatta verrebbe assegnata questa particolare etichetta di classe a quell'istanza di test.

2.3.2.2 Costruzione delle regole

Le regole di classificazione possono essere costruite secondo due differenti tecniche: metodo diretto e indiretto (o induttivo).

Utilizzando un **metodo diretto**, le regole vengono generate direttamente a partire dalla collezione dei dati di training. Tra gli algoritmi che utilizzano questo metodo ci sono RIPPER e CN2. Per la costruzione delle regole, utilizzando questo metodo, le istanze di training vengono analizzate ad una ad una e viene generata una regola. Quando viene creata una regola, l'istanza da cui viene generata viene eliminata. Dopo la generazione di tutte le regole ne viene fatta un'ottimizzazione.
Invece, utilizzando un metodo di tipo **indiretto** (o **induttivo**), le regole vengono estratte da un esistente modello di classificazione, generato utilizzando ad esempio un algoritmo classificazione basato sull' albero di decisione. L'algoritmo maggiormente utilizzato per la generazione delle regole secondo questa tecnica è il C4.5rules. Considerando quindi un modello di classificazione ad albero di decisione, in questa tipologia di estrazione delle regole, per ogni possibile percorso esistente tra il nodo radice e le foglie dell'albero viene creata una nuova regola. Come condizioni per una determinata regola vengono utilizzati tutti gli attributi di split attraversati in un percorso con i rispettivi valori, posti in AND logico tra loro. Come etichetta di classe della regola viene preso il valore del nodo foglia associato al percorso attraversato.
Considerando il modello di classificazione ad albero di decisione in Figura 2.7 si può presentare un esempio di generazione delle regole che utilizza una tecnica di costruzione indiretta:

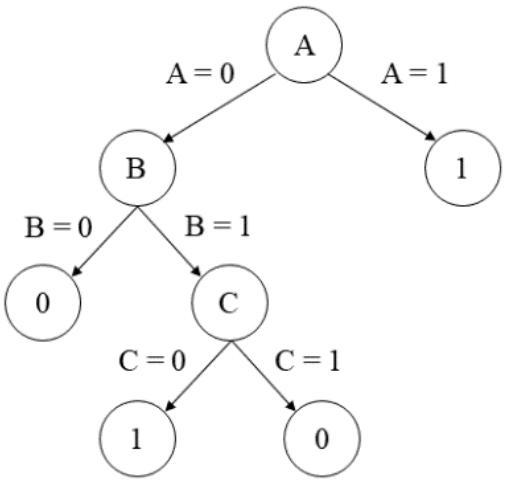


Figura 2.7 Albero di decisione da cui derivare le regole

Dall'albero di decisione è possibile generare quattro diverse regole, corrispondenti a tutti i percorsi esistenti tra il nodo radice e i nodi foglia. Le regole che verranno generate sono le seguenti:

R1: *A = 1 -> 1*
R2: *A = 0 AND B = 0 -> 0*
R3: *A = 0 AND B = 1 AND C = 0 -> 1*
R4: *A = 0 AND B = 1 AND C = 1 -> 0*

2.3.2.3 Qualità della classificazione basata su regole

In questo paragrafo verrà valutata le qualità della tecnica di classificazione basata su regole utilizzando i parametri di qualità discussi al Paragrafo 2.2.2.
Tra i punti di forza di una tecnica di classificazione di questo tipo troviamo:

- Accuratezza: risultati comparabili a quelli della classificazione tramite albero di decisione, anche se questa tipologia di classificazione presenta dei risultati leggermente superiori;
- Efficienza: regole generate facilmente e istanze classificate in modo rapido, poiché per la classificazione occorre semplicemente testare un insieme di regole;
- Interpretabilità: regole facili da interpretare;
- Scalabilità: comparabile ad un albero di decisione.

Invece, tra i punti di debolezza è presente:

- Bassa robustezza: come per l'albero di decisione, anche per la classificazione basata sulle regole l'accuratezza potrebbe diminuire in presenza di missing value.

2.3.3 Classificazione per regole di associazione

La classificazione associativa permette la classificazione di dati utilizzando regole estratte da coppie attributo-valore più frequenti, denominato **frequent itemset**, dalla collezione di dati da analizzare. [2] La creazione di un classificatore di questo tipo si basa su due principali fasi: ricerca delle coppie attributo-valore, chiamati **item**, più frequenti e successiva generazione delle regole di associazione. In un primo momento, si scorre tutta la collezione di dati considerando tutti gli item più frequenti, poi tra questi se ne selezionano soltanto un sottoinsieme che soddisfano dei criteri di qualità ed infine si generano le regole di associazione. I criteri di qualità che vengono utilizzati sono: la **confidenza** e il **supporto**.
La confidenza tra due valori degli attributi A e B è definita come la probabilità di trovare B in un'istanza avendo già trovato A.
Invece, il supporto è definito come il rapporto tra il numero di istanze che contengono sia A che B e il numero totale delle istanze nella collezione di dati. Questi parametri di qualità vengono calcolati per ogni coppia attributo-valore frequente e si selezionano soltanto quelle coppie che hanno, per questi parametri, dei valori al di sopra o almeno pari ad una determinata soglia definita. In questo modo, si considerano soltanto un sottoinsieme di coppie attributo-valore da cui si generano le regole di associazioni utilizzate per effettuare la classificazione. Per la selezione delle coppie attributo-valore più frequenti si parte dall'idea che data una collezione di dati da analizzare di dimensione N ne derivano 2^N itemset candidati. Per poter generare il frequent itemset si può adottare un approccio di tipo brute force, ma non è efficiente, quindi si cerca di adottare delle tecniche che consentano di ridurre il numero di itemset candidati alla creazione del frequent item set

finale. Per effettuare questa riduzione si utilizza l'algoritmo **FP-growth**, che sfrutta una rappresentazione ad albero dell'insieme di dati da analizzare chiamato **FP-tree**.

2.3.3.1 Qualità della classificazione per regole associative

La classificazione basata su regole associative permette di ottenere un elevato livello di accuratezza, in quanto si considerano le correlazioni tra i vari attributi della collezione di dati da analizzare. Inoltre, questa tecnica di classificazione è caratterizzata da un'alta scalabilità ed interpretabilità. Un classificatore di questo tipo presenta un'elevata robustezza, poiché le regole di associazione vengono create anche in mancanza di alcuni valori. Di contro, è poco scalabile su istanze che presentano un elevato numero di attributi ed è caratterizzato da una bassa efficienza; infatti, la generazione di buone regole associative è lunga e complessa poiché dipende dalla soglia di supporto scelta.

2.3.4 Reti neurali

Le reti neurali sono una tecnica di classificazione la cui struttura è composta da un insieme di nodi interconnessi tra loro che ne costituiscono l'unità computazionale. [2] All'interno di una rete neurale sono sempre presenti due tipologie di nodi: **nodi di input**, prendono in ingresso le istanze sia per la fase di apprendimento della rete che per quella di classificazione, e **nodi di output** che assegnano la classe all'istanza sotto analisi. Oltre a questi due tipi di nodi possono essere presenti anche degli altri, chiamati **nodi nascosti**, organizzati in vari livelli. Un modello di rete neurale è mostrato in Figura 2.8:

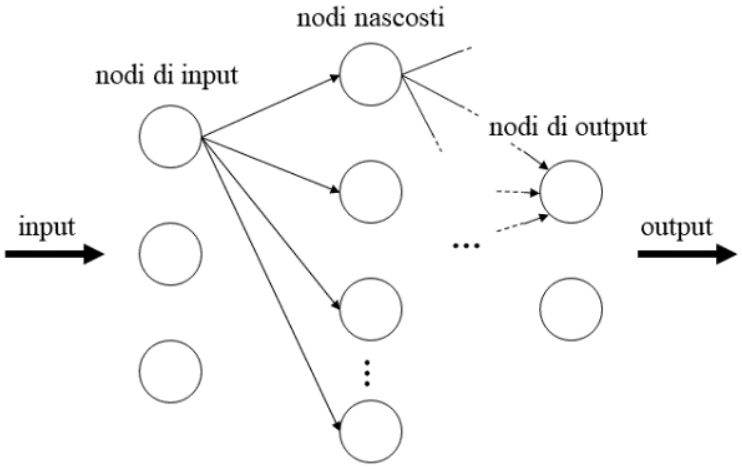


Figura 2.8 Modello di una rete neurale

In definitiva, una rete neurale è composta da vari livelli: da un primo livello costituito da un insieme di nodi di input, da uno o più livelli di insiemi di nodi nascosti e da un ultimo livello composto da un insieme di nodi di output.
In un modello a rete neurale sono associate altre due caratteristiche: pesi e offset. Ad ogni connessione tra i vari nodi che formano una rete neurale è associato un **peso** che inizialmente assume un valore casuale, ma durante la fase di apprendimento della rete viene continuamente aggiornato a seguito di un errore di classificazione di un'istanza rispetto al valore di classe previsto. Inoltre, ad ogni nodo è associato anche un valore di **offset**.

2.3.4.1 Funzionamento di una rete neurale

In una rete neurale gli input inviati ai nodi di input sono costituiti dal valore degli attributi dell'istanza che deve essere analizzata. L'output di questo primo livello della rete rimane invariato, poiché in uscita dai nodi di input sono presenti gli stessi valori che vengono forniti in input. In ogni nodo appartenente ai livelli successivi al primo, nodi nascosti e nodi di output, avviene l'effettiva computazione. Infatti, gli input di questi livelli corrispondono agli output dei livelli precedenti in cui però bisogna considerare il peso associato al collegamento tra i due nodi ed un valore caratteristico del nodo, l'offset. Considerando un nodo n tra i nodi nascosti o tra quelli di output, il suo input I_n è dato dalla seguente relazione:

$$I_n = \sum_i w_{i,n} O_i + offset_n$$

dove $w_{i,n}$ è il peso del collegamento tra il nodo i del livello precedente e il nodo n preso in considerazione, O_i è l'output del nodo i del livello precedente e $offset_n$ è l'offset associato al nodo n considerato.
Ogni nodo, inoltre, applica poi una **funzione di attivazione** sul valore che riceve in input ed invia il suo output al livello successivo. Infine, quando viene generato l'output dai nodi di output, se durante la fase di apprendimento si verifica un errore tra il valore della classe calcolato e quello previsto per un'istanza, viene calcolato l'errore da ogni nodo di output e viene propagato ai livelli precedenti, dove vengono sistemati i valori dei pesi e degli offset di tutti i nodi di tutti i livelli che costituiscono la rete neurale.

2.3.4.2 Qualità della classificazione tramite rete neurale

Tra tutte le diverse tecniche di classificazione, le reti neurali sono tra quelle con il più alto livello di accuratezza. Per raggiungere tale livello di accuratezza occorre un tempo di training della rete neurale lungo e una complessa configurazione di tutti i parametri. Quindi, i vantaggi di una classificazione tramite rete neurale sono un'elevata accuratezza della classificazione ed un'alta robustezza ai rumori e ai missing value, poiché i dati vengono propagati più volte nella rete ed ogni iterazione questi vengono riassorbiti.

Inoltre, questa tecnica di classificazione consente di effettuare la regressione, in quanto è in grado di generare output sia di tipo discreto che continuo. Di contro, gli svantaggi sono bassa efficienza, poiché è necessario un tempo di apprendimento lungo ed una complessa configurazione dato che occorre scegliere in modo accurato parametri, e la non interpretabilità del modello, poiché sono presenti soltanto un insieme di pesi ed un offset per ogni nodo. La tecnica di classificazione basata su rete neurale, data la non interpretabilità del modello, viene utilizzata quando si vuole raggiungere un'elevata accuratezza, ma non è necessario sapere come è stata presa la decisione della classificazione.

2.3.5 Classificazione Bayesiana

Un classificatore Bayesiano consente la classificazione di collezioni di dati utilizzando dei valori di probabilità ricavati grazie all'applicazione del **teorema di Bayes**. [2] Il teorema di Bayes permette di calcolare la probabilità che una data istanza appartenga ad una determinata classe, indicata come $P(I|C)$, dove I indica l'istanza considerata e C è il valore di una determinata classe. Tra tutti i valori di probabilità che vengono calcolati per una determinata istanza da classificare viene preso il valore massimo; questo indica che quell'istanza ha maggiore probabilità di appartenere a quella classe scelta in confronto a tutte gli altri possibili valori delle classi. Dal teorema di Bayes la probabilità $P(I|C)$ è data dalla formula seguente:

$$P(C|I) = \frac{P(I|C)P(C)}{P(I)}$$

dove $P(I)$ è un valore costante per tutti i valori di C, quindi è trascurabile, $P(C)$ è chiamata **probabilità a priori** e $P(C|I)$ è chiamata **probabilità a posteriori**. Il calcolo di $P(I|C)$ è effettuato grazie all'assunzione dell'**ipotesi Naïve** con la quale, data un'etichetta di classe, gli attributi che compongono l'istanza sono considerati come statisticamente indipendenti tra loro. Tuttavia, questa ipotesi non è sempre vera ed è la ragione per cui ne influenza l'accuratezza della classificazione. Quindi, è possibile effettuare il calcolo della probabilità considerando la seguente assunzione:

$$P(I|C) = \prod_{i=1}^{k} P(a_i|C)$$

Dove k è il numero di attributi dell'istanza e a_i è il valore dell'attributo i-esimo.
In definitiva, per l'assegnazione dell'etichetta di classe adb un'istanza occorre il calcolo di tutte le probabilità per cui quell'istanza appartenga alle varie classi $P(C_i|I)$ e la successiva selezione del valore dell'etichetta di classe che presenta il valore di probabilità più alto.

2.3.5.1 Qualità di un classificatore Bayesiano

Un classificatore Bayesiano presenta un'elevata efficienza paragonabile alla classificazione basata sugli alberi di decisione, poiché occorre soltanto calcolare tutte le probabilità e scegliere la massima tra queste. Inoltre, è una tecnica di classificazione incrementale, caratteristica che non è presente in nessun'altra tecnica precedentemente analizzata. Presenta questa caratteristica poiché, in presenza di una nuova istanza di training, occorre ritoccare soltanto qualche numeratore e denominatore nelle formule per la successiva classificazione. In aggiunta, una volta costruito il modello, non sono più necessari i dati di training. Questa tecnica di classificazione è incrementale anche senza l'utilizzo del training set, ma considerando solo il modello. Tra i difetti che presenta un classificatore Bayesiano sono presenti la non interpretabilità, poiché è possibile interpretare solo il singolo valore di probabilità e dire che quello ha avuto un peso maggiore nella classificazione, e la limitata accuratezza, poiché è fortemente influenzata dall'ipotesi Naïve.

2.3.6 Reti Bayesiane

Le reti Bayesiane consentono di classificazione collezioni di dati trascurando l'ipotesi di Naïve. [2] Questa ipotesi, come detto precedentemente, non è sempre vera sui dati e ne limita l'accuratezza della classificazione. Le reti Bayesiane, inoltre, considerano le variabili come divise in due diversi sottoinsiemi: variabili dipendenti tra loro e non.

2.3.6.1 Struttura di una rete bayesiana

Una rete bayesiana è costituita da due principali componenti: un **DAG**, directed acyclic graph, e una **tabella CPT** per ogni attributo, tabella di probabilità condizionale.
In un DAG, ogni nodo è rappresentato da un attributo preso dalla collezione dei dati di training e ogni arco rappresenta una relazione tra i due nodi collegati. Due nodi connessi da un arco sono dipendenti, invece, quelli non collegati tra loro sono statisticamente indipendenti.
Una tabella CPT, invece, rappresenta tutti i valori di probabilità tra un determinato nodo della rete e tutti i suoi predecessori. Infatti, per tutti i valori che possono assumere i nodi predecessori, che corrispondono agli attributi di una collezione di dati, elenca tutti i valori di probabilità per cui questi implicano un determinato valore dell'attributo del nodo preso in considerazione.

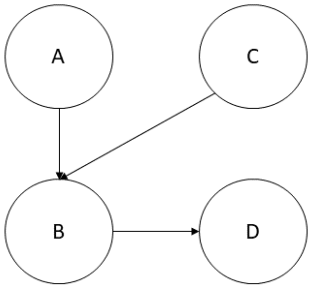


Figura 2.9 DAG di una rete bayesiana

Nella Figura 2.9 è rappresentato il DAG di una rete bayesiana di esempio costituita da quattro nodi, A, B, C, D, e da tre archi che collegano A e C con B e B con D. Secondo le definizioni enunciate in precedenza, A e C sono i predecessori di B, quindi il nodo B è statisticamente dipendente dai nodi A e C. Invece, il nodo D è statisticamente indipendente dai nodi A e C, ma è dipendente dal nodo B, suo predecessore.
Una CRT della rete bayesiana di esempio mostrata in Figura 2.9 può essere quella rappresentata nella Tabella 2.1:

	A and C	!A and C	A and !C	!A and !C
B	0.4	0.75	0.35	0.5
!B	0.6	0.25	0.66	0.5

Tabella 2.1: Esempio di tabella di probabilità condizionale di una rete bayesiana

2.3.7 Support vector machine

Una tecnica di classificazione basata su **support vector machine (SVM)** consente di classificare collezioni di dati sia lineari che non lineari. [2] Una SVM rappresenta tutte le istanze della collezione dei dati di training su un piano formato da un numero di assi (dimensioni) pari al numero degli attributi che costituiscono le istanze. Ad esempio se un'istanza è costituita da tre attributi, allora i dati di training saranno rappresentati su un piano tridimensionale. Le tre principali caratteristiche di un classificatore SVM sono: **linee** o **iperpiani**, a seconda se il classificatore presenta, rispettivamente, un grafico bidimensionale o n-dimensionale, **margini** e **vettori di supporto**. Una linea o un iperpiano costituisce un "confine" che permette di classificare le istanze appartenenti a classi diverse dividendole tra loro. Un margine è una distanza tra le due istanze di classi diverse più vicine. Invece, i vettori di supporto corrispondono alle istanze più difficili da classificare per una SVM, poiché sono quelle che si trovano all'interno dei margini di un iperpiano.

2.3.7.1 Tipologie di support vector machine

Una SVM permette sia la classificazione collezioni di dati linearmente separabili tra le varie classi sia quelle le cui istanze sono separabili in maniera non lineare. Le istanze di una collezione di dati sono linearmente separabili se, dopo averle rappresentate su un piano, sono separabili da una o più linee rette. Nella Figura 2.10 è rappresentato un esempio di grafico bidimensionale in cui sono raffigurate delle istanze di training linearmente separabili.

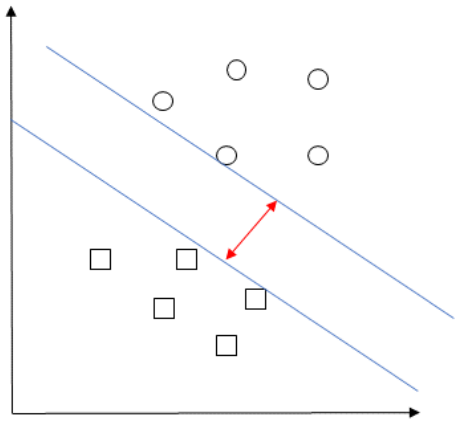


Figura 2.10 SVM lineare

I dati di training raffigurati nella Figura 2.10 sono costituiti da dieci istanze appartenenti a due classi differenti. Queste istanze sono separabili da linee rette, quindi risultano separabili in maniera lineare.
Di contro, quando le istanze di una collezione di dati non sono separabili da linee rette, sono separabili in maniera non lineare. Un esempio è mostrato nella Figura 2.11 in cui non è possibile trovare una linea retta che separi le istanze rappresentate appartenenti a due classi differenti.

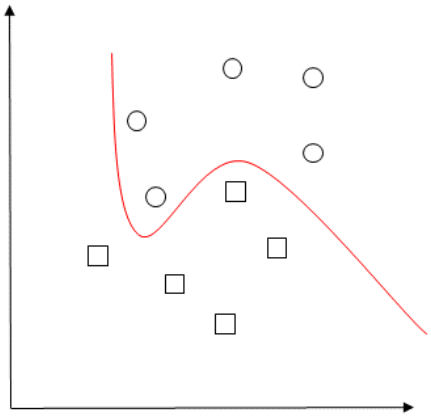


Figura 2.11 SVM non lineare

Le istanze di training di esempio mostrate nella Figura 2.11 sono separabili dalla linea rossa, ma dato che questa non è una retta, le istanze non sono separabili in maniera lineare.

2.3.7.2 Funzionamento di una SVM

Una SVM si applica in maniera differente a seconda della tipologia delle istanze di una collezione di dati da classificare, linearmente separabili e non.
In presenza di istanze separabili in maniera lineare, occorre trovare tra tutte le rette o tra tutti gli iperpiani che le separano tra le diverse classi quelle che massimizzano il valore del margine. Infatti, viene selezionata la retta o l'iperpiano con valore di margine massimo, poiché consente di minimizzare l'errore di classificazione. Un esempio che rappresenta due iperpiani caratterizzati due diversi valori di margine è mostrato nella Figura 2.12:

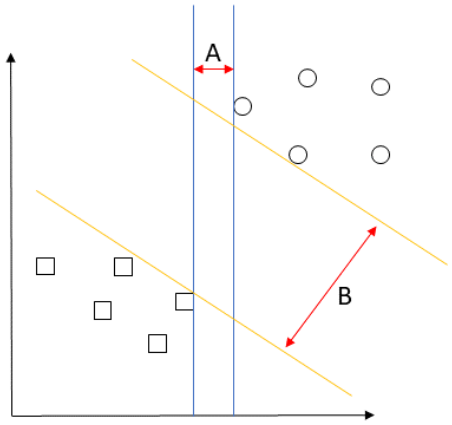


Figura 2.12 Esempio SVM lineare

Come si evince dalla Figura 2.12, i due iperpiani presentano margini di diversa grandezza ed in questo caso verrà selezionato l'iperpiano con margine B > A.
Per quanto riguarda collezioni di dati separabili in maniera non lineare il procedimento di classificazione è più complesso, in quanto occorre operare in due fasi separate ed estendere il comportamento del precedente approccio. Nella prima fase le istanze della collezione di dati vengono mappate su uno spazio dimensionale più grande in modo da renderle separabili in maniera lineare. Infine, nella seconda fase è possibile utilizzare l'approccio precedente di ricerca una linea o un iperpiano che massimizza la grandezza del margine, dato che adesso le istanze sono linearmente separabili.

2.3.7.3 Qualità della classificazione tramite SVM

Un classificatore basato su support vector machine tra i vantaggi presenta la capacità di risolvere problemi di classificazione difficili e non lineari e di garantirne un alto livello di accuratezza di classificazione. Per quanto riguarda la risoluzione di problemi semplici, invece, l'accuratezza è paragonabile a quella di una tecnica di classificazione basata su un albero di decisione. Tra gli svantaggi sono presenti l'elevato tempo di creazione del modello, inferiore a quello impiegato da una rete neurale, e la non interpretabilità del modello.

2.3.8 K-Nearest Neighbours

La tecnica di classificazione di tipo **k-Nearest Neighbour (k-NN)** non prevede la creazione di un modello di classificazione per etichettare le istanze di test, ma conserva in un grafico le istanze di training e successivamente, utilizzando proprio questo grafico, riesce ad effettuare la classificazione delle istanze di test. [2] Il grafico utilizzato per il salvataggio delle istanze di training è un grafico *n*-dimensionale, in cui *n* corrisponde al numero di attributi che caratterizzano le istanze appartenenti alla collezione dei dati di training. Nel grafico le istanze vengono rappresentate come dei punti. La classificazione, secondo questa tecnica, avviene rappresentando l'istanza di test nel grafico e scegliendo i k punti più vicini a questa. All'istanza di test viene assegnata il valore della classe più presente tra quelle delle k istanze di training più vicine.

2.3.8.1 Calcolo della distanza tra i vicini

Per effettuare in modo corretto il calcolo della distanza occorre, innanzitutto, normalizzare il valore degli attributi, diversamente si potrebbero preferire degli attributi rispetto ad altri. Il calcolo della distanza tra attributi con valori numerici viene effettuato utilizzando il **teorema della distanza Euclidea tra due punti** enunciato dalla formula seguente:

$$d(x, y) = \sqrt[2]{\sum_i (x_i - y_i)^2}$$

dove $d(x, y)$ è il valore della distanza tra due punti x e y.
Invece, per il calcolo di attributi con valori non numerici, ad esempio gli attributi di tipo categorico, si procede in un altro modo: considerate due istanze, se i due valori di un attributo coincidono, allora sarà dato un valore di distanza 0, altrimenti sarà assegnato il valore della distanza massimo preso dall'intervallo di normalizzazione per quell'attributo.

2.3.8.2 Selezione del parametro k

Nella classificazione di tipo k-Nearest Neighbour è di fondamentale importanza la giusta selezione del parametro k, poiché ne influenza l'accuratezza della classificazione. Se si scegliesse un valore di k troppo basso, la classificazione potrebbe essere influenzata da istanze di rumore.

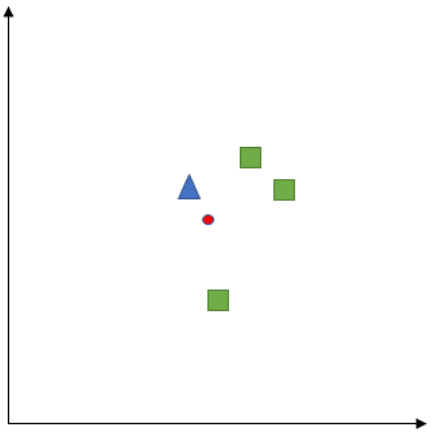


Figura 2.13 Classificatore k-NN

Nella Figura 2.13 è mostrato un grafico bidimensionale di esempio in cui sono rappresentate quattro istanze di training, una blu e tre verdi, e un'istanza di test rossa. Se venisse scelto un valore del parametro k = 1, l'istanza di test verrebbe etichettata con la classe blu erroneamente, perché per k = 3 o k = 4 sarebbe stata selezionata l'etichetta di classe corretta, la verde.
Di contro, un valore di k troppo grande potrebbe portare all'inclusione di punti appartenenti ad altre classi con la possibilità di provocare un errore di classificazione.
In definitiva, la selezione del parametro k gioca un ruolo chiave nella tecnica di classificazione k-NN.

2.3.8.3 Funzionamento del classificatore k-NN

La tecnica di classificazione k-NN prevede l'iniziale rappresentazione tramite punti di tutte le istanze di training su un grafico il cui numero di assi è pari al numero di attributi delle istanze della collezione dei dati di training. Infatti, per la fase di apprendimento, questa tecnica di classificazione deve soltanto memorizzare in maniera efficiente i dati di training per la successiva lettura. In seguito, viene scelto il valore del parametro k. Quando occorre classificare un'istanza di test, anch'essa viene rappresentata su questo grafico e ne viene calcolata la distanza tra il punto corrispondente all'istanza di test con tutti i punti rappresentanti le istanze di training. Tra tutti questi valori di distanza vengono selezionati i k valori più bassi. In aggiunta al calcolo della distanza, nella selezione delle

k istanze, viene utilizzato un valore di peso inversamente proporzionale al quadrato della distanza, in modo da dare maggiore peso alle istanze di training più vicine al punto che rappresenta l'istanza di test. Dopo aver selezionato le k istanze di training più vicine, viene scelto il valore di classe più presente tra le k istanze selezionate e viene assegnato all'istanza di test.

2.3.8.4 Qualità di un classificatore k-NN

Una classificazione effettuata tramite la tecnica k-NN presenta un buon valore del livello accuratezza e consente la gestione anche di casi particolari di una collezione di dati, poiché conserva tutti i dati di training. Quindi, conservando tutto il training set non fa alcuna astrazione delle istanze presenti ed in questo modo riesce a considerare ogni caso particolare. Come un classificatore bayesiano, anche la tecnica di classificazione k-NN presenta la caratteristica di essere incrementale, poiché in presenza di un nuovo dato etichettato basta aggiungerlo tra i dati di training precedentemente presenti nel grafico.
Uno degli svantaggi di questa tecnica di classificazione è l'elevato tempo di classificazione, in quanto occorre calcolare tutte le distanze tra i punti del training set e tutti i nuovi punti che devono essere etichettati. Maggiore è la cardinalità del training set, maggiore sarà questo tempo di classificazione. Quindi, un classificatore di questo tipo va utilizzato per la risoluzione di problemi di classificazione in cui non occorre velocità. Inoltre, un altro svantaggio è la non interpretabilità, in quando questo classificatore non presenta alcun modello, ma soltanto tutte le istanze del training set.

2.4 Descrizione e analisi degli algoritmi di classificazione per Big data

In questo paragrafo, dopo un'iniziale definizione del concetto dei Big Data, verrà presentata la problematica relativa all'analisi di questa particolare tipologia di dati. Infine, verranno descritte le principali tecniche di classificazione disponibili per i Big Data.

2.4.1 Big Data

Con il termine **Big Data** si identificano collezioni di dati di grandi dimensioni che presentano strutture complesse e caratteristiche differenti dai normali dati. [3] Questi dati vengono generati da diverse sorgenti e presentano strutture e formati differenti. È possibile trovare immagini, dati di basi di dati relazionali e non relazionali, ed occorre unirli anche se presentano formati diversi. Da un paio d'anni a questa parte con l'aumento dell'utilizzo dei social network e l'avvento dell'internet of things (IoT) milioni e milioni di dati vengono generati e trasmessi sul Web, ma questi sono soltanto due delle possibili fonti dei cosiddetti Big Data. I Big Data sono caratterizzati dalle "tre V":

- **Volume**: incrementa sempre più in modo esponenziale la cardinalità delle collezioni di dati;
- **Varietà**: vengono generati dati di qualsiasi tipo e struttura;
- **Velocità**: si producono dati sempre più velocemente.

Oltre a queste tre caratteristiche che contraddistinguono i Big Data dai dati strutturati non big, ne sono presenti altre due: **veracity** e **value**.
Il termine veracity si riferisce al livello di affidabilità delle collezioni di dati in termini di qualità, poiché dati di così grandi dimensioni e varietà sono costituiti dall'elevata presenza di rumori.
In aggiunta, i Big Data presentano come caratteristica il termine value, perché occorre analizzare i dati per arrivare ad estrarre informazioni utili. Da notare che quest'ultima caratteristica è la più importante.
Come per i dati strutturati, anche sui Big Data vengono utilizzate delle tecniche di analisi per l'estrazione di informazioni utili. Purtroppo, non è possibile analizzare i dati costituiti da queste caratteristiche tramite le normali tecniche ed architetture utilizzate dai dati non big, poiché non sono in grado di salvarli e gestirli in modo efficiente. Le normali tecniche di analisi di dati strutturati utilizzano un approccio in cui i dati vengono caricati nella RAM di un'unica unità elaborativa e successivamente analizzati. Data l'impossibilità di utilizzo di questo approccio a causa dell'elevato volume che caratterizza questa tipologia di dati, occorre adottare delle architetture distribuite in cui i dati vengono divisi in piccole porzioni ed analizzate una per volta da diverse unità di elaborazione per poi combinarne il risultato finale dell'analisi. Occorre notare, inoltre, che le applicazioni di analisi per Big Data sono maggiormente rivolte ad una analisi di tipo offline.

2.4.2 Analisi dei Big Data

L'utilizzo di tecniche di analisi sui Big Data consente un processo di estrazione di informazioni utili ed è di fondamentale importanza per le aziende a cui interessano, in particolare, analisi di tipo **descrittive** o **predittive**. Grazie ad un'analisi descrittiva un'azienda può rendersi conto dell'andamento attuale delle sue performance, invece, utilizzando tecniche di tipo predittive riesce a predire cosa potrebbe avvenire in futuro utilizzando la conoscenza di tutti i dati del passato. Occorre prestare attenzione alla qualità della collezione di dati da sottoporre ad analisi, poiché si potrebbe incorrere ad errori di valutazione a causa dell'elevata varietà che caratterizza i Big Data.
Quando una collezione di dati presenta le tre caratteristiche proprie dei Big Data, volume, varietà e velocità, risulta impossibile l'utilizzo delle normali tecniche di analisi. Quindi, per poter procedere all'analisi di dati con queste caratteristiche occorre utilizzare delle tecniche ed architetture che scalano in accordo alla cardinalità di queste collezioni di dati. Questo approccio porta a benefici quali maggior velocità di analisi e più alta efficienza. Per la conservazione e l'analisi di dati di questo tipo vengono utilizzati dei **data center**, costituiti da **cluster** formati da un elevato numero di unità elaborative, chiamati **nodi**, che consentono di far fronte all'elevata cardinalità dei Big Data. In un'architettura di questo

tipo le collezioni di dati vengono divise in porzioni e ogni nodo ne analizza una di queste fornendo un risultato parziale che, successivamente, viene combinato con i risultati di tutti gli altri nodi per generare il risultato finale dell'analisi. Viene utilizzato un tale approccio poiché, data l'elevata cardinalità delle collezioni di dati, risulterebbe non efficiente conservare i dati in nodi diversi rispetto a dove verranno analizzati in quanto dovrebbero essere continuamente inviati verso le unità elaborative per poi essere analizzati.
Per poter effettuare l'analisi dei Big Data vengono comunemente usati dei framework come **Spark**, poiché consentono di fare fronte a tante problematiche legate alla gestione dei nodi dei cluster ed all'elaborazione di dati di così elevata cardinalità, come ad esempio la gestione dei guasti dei nodi del sistema.
Occorre prestare attenzione al fatto che le tecniche di analisi per Big Data non presentano un'elevata accuratezza in confronto alle normali tecniche di Data Mining, quindi prima di procedere all'analisi di una collezione di dati utilizzando questo approccio occorre essere sicuri di essere in presenza di dati le cui caratteristiche sono quelle dei Big Data. Infatti, nel caso contrario, l'utilizzo di un approccio non distribuito e di tecniche Data Mining risulterebbe più efficiente e si otterrebbe un maggiore livello di accuratezza nell'analisi.

2.4.3 Spark

Spark è un framework open-source che consente di effettuare in modo efficiente l'analisi dei Big Data. Tra le diverse caratteristiche consente di avere una bassa latenza, alto parallelismo e permette di gestire gli eventuali guasti che possono avvenire nel sistema. Per consentire l'analisi di collezioni di dati utilizza un approccio di tipo iterativo usando la RAM dei nodi di elaborazione o anche il disco fisso se la dimensione dei dati supera quella della RAM. Questo approccio consente di aumentare notevolmente la velocità di elaborazione e l'efficienza rispetto ad altri framework che utilizzando invece soltanto il disco. I dati vengono letti dal sistema distribuito di archiviazione soltanto una volta e vengono caricati in RAM utilizzando un'astrazione, i cosiddetti **Resilient Distributed Datasets (RDDs)**. Gli RDD sono una collezione di oggetti distribuiti tra le RAM dei nodi dei cluster di un data center che rappresentano la collezione di dati. Tutte le operazioni in Spark avvengono utilizzando questa astrazione. Infatti, per l'analisi dei dati, Spark utilizza due principali insiemi di operazioni applicabili sugli RDDs: **trasformazioni** e **azioni**.
Le trasformazioni sono operazioni che hanno come input un RDD e come output un RDD differente, poiché gli RDD sono immutabili, e permettono di effettuare una determinata operazione sui dati di un RDD di input. Un esempio di trasformazione applicabile su un RDD è la trasformazione denominata *filter*, che consente di filtrare i dati che presentano determinate caratteristiche.
Le azioni, invece, sono operazioni che permettono di ottenere un risultato in locale, quindi che non viene salvato nel sistema di archiviazione distribuito. Infatti, il risultato di

un'azione non è più sotto forma di RDD. Un esempio di azione è quella denominata *count*, che consente di ottenere il numero di elementi che possiede un RDD.
Un componente di fondamentale importanza in un'applicazione Spark è il **Driver**, costituito da un programma scritto in un linguaggio di programmazione che gira sul nodo di elaborazione principale, il **master node**. Infatti, questo componente consente di creare, a partire dalla collezione di dati, gli RDDs che vengono distribuiti tra i nodi dei cluster ed applicare su di essi trasformazioni ed azioni. Inoltre, descrive il flusso delle operazioni da eseguire sugli RDDs e permette di comunicare con tutti i nodi di elaborazione, detti **worker node**, e di coordinarli nell'esecuzione in parallelo di tali operazioni sulla porzione di RDD ad essi associata. Infine, consente di aggregare tutti i risultati ottenuti dai worker node e generare, quindi, il risultato finale dell'analisi.
Spark, inoltre, è costituito da un insieme di componenti, ognuno dei quali fornisce una specifica funzionalità. Il principale componente è lo **Spark Core** che presenta le funzionalità essenziali per il funzionamento di tutto il framework, come la gestione della memoria e dei guasti, la divisione dei task nei vari nodi di elaborazione e il monitoraggio di tutti questi task. Altri due componenti fondamentali sono: il componente **Spark Streaming**, che permette di effettuare un'analisi real-time su uno stream di dati, e la **libreria MLib** (**machine learning library**), che fornisce le API per l'utilizzo degli algoritmi utili all'analisi dei Big Data.

2.4.4 Tecniche di classificazione disponibili in Spark

Non tutte le tecniche di classificazione descritte ed analizzate nel Paragrafo 2.3 possono essere utilizzate per i Big Data, poiché per questa tipologia di dati occorre utilizzare degli algoritmi parallelizzabili, dato che è necessario un approccio di tipo distribuito. Tra gli algoritmi del Paragrafo 2.3 solo alcuni sono parallelizzabili e quindi utilizzabili anche in contesti Big Data. Nella libreria MLib di Spark sono presenti tecniche di classificazione quali: alberi di decisione, classificatori bayesiani e support vector machine lineari. Tutte queste tecniche hanno in comune alcuni concetti propri del framework Spark quali: DataFrame, Trasformer, Estimator, Pipeline e Parameter.
I **DataFrame** consentono di effettuare un'astrazione della collezione di dati da sottoporre a classificazione. Un DataFrame può rappresentare dati di diverso tipo in colonne.
Un **Trasformer**, invece, consente di trasformare i DataFrame in input, tramite l'utilizzo di un algoritmo, in DataFrame di output. Un esempio di Trasformer è un modello di classificazione.
Gli **Estimator** vengono utilizzati per la creazione dei Trasformer grazie all'utilizzo dei DataFrame in input. Un esempio di Estimator è la tecnica di classificazione ad albero di decisione.
Un insieme di Trasformer ed Estimator costituiscono la cosiddetta **Pipeline**, che rappresenta il flusso di esecuzione e di analisi su una collezione di dati in input.
Infine, i **Parameter** costituiscono i parametri passati agli Estimator ed ai Trasformer.
Dati questi concetti, ogni algoritmo di classificazione della libreria MLib di Spark consente di creare una pipeline formata da diversi Trasformer ed Estimator, a seconda del

tipo di tecnica di classificazione che si vuole applicare. La pipeline è utilizzata per la generazione del modello utile alla successiva classificazione di dati di test.

2.4.4.1 Alberi di decisione per Big Data

La tecnica di classificazione basata sull'albero di decisione, anche in contesti Big Data, si basa sulla costruzione di un modello di classificazione ad albero, dove ogni nodo è costituito da un attributo della collezione di dati scelto in maniera greedy utilizzando gli indici di qualità descritti nel Paragrafo 2.3.1.4. La libreria MLib di Spark supporta alberi di decisione che permettono una classificazione sia binaria che multi-classe. Quando si utilizza una tecnica ad albero di decisione in Spark, la collezione di dati iniziale di grandi dimensioni è distribuita su diversi nodi di elaborazione (worker node) grazie all'astrazione data dagli RDD, introdotti nel Paragrafo 2.4.3.
Per la costruzione del modello di classificazione si parte inizialmente dalla scelta del nodo radice dell'albero di decisione. Ogni worker node, in parallelo, calcola delle statistiche sui possibili attributi su cui effettuare lo split in base alla porzione di dati ad esso associato. Le statistiche di tutti i nodi di elaborazione vengono inviate al Driver che le aggrega e decide l'attributo migliore su cui effettuare lo split e lo comunica ai worker node. Questo approccio viene applicato in maniera ricorsiva per la costruzione del modello di classificazione fino al raggiungimento di una condizione di fine.

2.4.4.2 Classificatori Bayesiani per Big Data

Nella libreria MLib di Spark è presente anche la tecnica di classificazione bayesiana basata sull'ipotesi Naive, la quale data un'etichetta di classe assume che gli attributi che compongono l'istanza sono statisticamente indipendenti tra loro. Questa tecnica di classificazione utilizza un approccio di calcolo della probabilità che una data istanza appartenga ad una data classe. In un'architettura distribuita, quindi, ogni nodo di elaborazione, basandosi sui valori di probabilità globali, classifica la porzione di dati ad esso assegnata.

2.4.4.3 SVM lineari per Big Data

La tecnica di classificazione denominata support vector machine per architetture non distribuite è in grado di classificare collezioni di dati sia lineari che non lineari, come spiegato al Paragrafo 2.3.7. Nella libreria MLib di Spark è presente soltanto la tecnica support vector machine lineare, poiché viene utilizzato un approccio distribuito e, quindi, è necessario una tecnica di classificazione parallelizzabile.

Capitolo 3
Tecnica di compressione di dati mirata alla classificazione

In questo capitolo, nella prima parte, verrà descritto ad alto livello l'algoritmo realizzato durante il percorso di tesi. Nella seconda parte del capitolo, invece, ne verrà analizzata nel dettaglio anche l'implementazione.

3.1 Descrizione del lavoro svolto

Date le problematiche legate alla classificazione di grandi moli di dati analizzate nel Paragrafo 1.1, durante il percorso di tesi è stato realizzato un algoritmo per la compressione di dati etichettati finalizzato alla classificazione di grandi moli di dati. L'algoritmo proposto è in grado di generare una versione compressa della collezione di dati di partenza il cui modello di classificazione generato da esso porta ad una qualità di classificazione simile a quella ottenuta dal modello costruito dalla collezione di dati iniziale di grande dimensione.
L'algoritmo implementato in questa tesi è suddivisibile in varie fasi. La prima fase dell'algoritmo è costituita dalla fase di compressione della collezione di dati di input. Grazie all'utilizzo di una nota tecnica di classificazione basata sull'albero di decisione, inizialmente viene generato un modello di classificazione utilizzando come dati di training la collezione di dati di partenza che viene utilizzato per effettuare la compressione. La compressione avviene classificando, tramite il modello di classificazione generato in precedenza, dei dati di test che corrispondono alla collezione di dati di partenza. Per classificare le istanze di test è stata modificata la funzione di classificazione dell'algoritmo scelto. L'algoritmo implementato permette di dividere le istanze etichettate dal modello in due diverse tipologie: quelle classificate correttamente dal classificatore e quelle classificate in modo errato. È possibile rendersi conto se un'istanza è stata classificata correttamente o no confrontando le etichette di classe assegnate dal modello di classificazione con quelle di partenza, in quanto i dati di test erano già etichettati. La modifica apportata alla funzione di classificazione standard dell'algoritmo basato sull'albero di decisione consente di tenere invariate le istanze erroneamente classificate, invece, di quelle classificate correttamente ne viene conservato il percorso che queste attraversano nel modello utilizzato per la compressione. Per ogni percorso, inoltre, viene tenuto conto di un peso che corrisponde al numero di volte in cui questo è stato attraversato da un'istanza correttamente classificata. Da notare che l'algoritmo originale di classificazione non tiene conto dei percorsi attraversati dalle istanze durante la classificazione, quindi questo è stato opportunamente modificato. In seguito, i percorsi ottenuti vengono ritrasformati in istanze e la versione compressa della

collezione di dati iniziale sarà costituta dall'unione di queste nuove istanze estratte dai percorsi e dalle istanze classificate in modo errato mantenute invariate. Per spiegare meglio tutto il flusso di esecuzione dell'algoritmo realizzato e le varie fasi di funzionamento, un diagramma a blocchi è mostrato nella Figura 3.1:

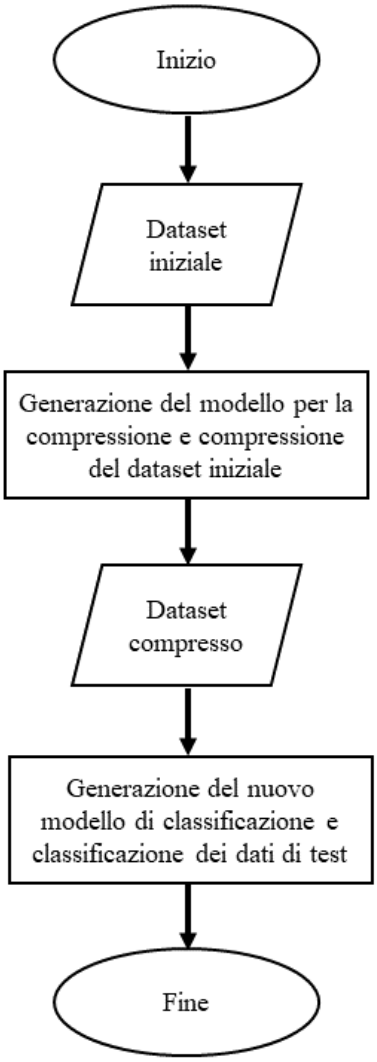


Figura 3.1 Diagramma a blocchi del funzionamento dell'algoritmo

Dal diagramma a blocchi della Figura 3.1 si evince che dopo una fase preliminare in cui viene utilizzata la collezione di dati iniziale per la generazione del modello utile alla compressione, dopo la fase di compressione viene poi utilizzata la versione compressa della collezione di dati iniziale per la costruzione del nuovo modello di classificazione. Questo modello verrà utilizzato per l'effettiva classificazione delle istanze di test.

Considerando le cinque istanze di esempio della Tabella 3.1 ed il modello di classificazione ad albero di decisione di esempio mostrato in Figura 3.2 è possibile spiegare il funzionamento della classificazione volta alla compressione utilizzata dall'algoritmo implementato in questa tesi. Le istanze di test di esempio rappresentate nella Tabella 3.1 sono costituite da quattro attributi, di cui quello denominato "etichetta di classe" ne costituisce l'attributo di classe.

attributo-1	**attributo-2**	**attributo-3**	**etichetta di classe**
0	0	1	A
0	1	1	B
0	1	1	B
0	1	0	B
0	0	0	B

Tabella 3.1: Istanze etichettate di una collezione di dati

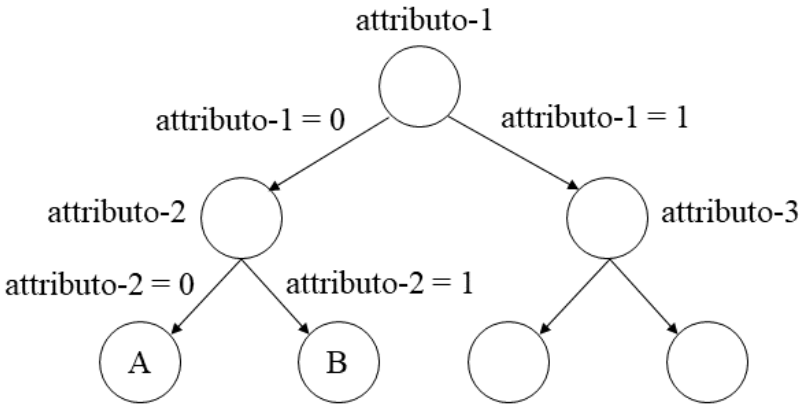


Figura 3.2 Modello di compressione ad albero

Quando le istanze della Tabella 3.1 vengono analizzate dal modello di classificazione utilizzato per la compressione mostrato in Figura 3.2, le prime quattro vengono classificate correttamente e, grazie alla modifica apportata alla funzione di classificazione nell'algoritmo implementato, ne viene salvato il percorso che queste hanno attraversato nel modello ed un fattore di peso. Il peso associato ad ogni percorso rappresenta il numero di volte in cui quel percorso è stato attraversato da un'istanza correttamente classificata. Come risultato della funzione di classificazione modificata si ottengono, quindi, i due percorsi mostrati di seguito:

attributo-1 = 0, attributo-2 = 0, classe = A – peso = 1
attributo-1 = 0, attributo-2 = 1, classe = B – peso = 3

Come si nota dalle informazioni sui percorsi effettuati dalle prime quattro istanze classificate correttamente, si nota che tutte queste hanno attraversato il ramo di sinistra dell'albero di decisione dopo aver effettuato un test sull'attributo-1, che ne costituisce la radice del modello ad albero di esempio della Figura 3.2. Successivamente, si passa al test sull'attributo-2 e solo un'istanza su quattro ha attraversato il ramo di sinistra ed è stata classificata con l'etichetta di classe A, invece, le restanti tre hanno percorso il ramo di destra e sono state etichettate con B. Da notare che in queste istanze classificate in modo corretto e successivamente trasformate in percorsi non è più necessario il valore dell'attributo-3.
L'ultima istanza di test, invece, viene erroneamente classificata dal modello, quindi viene salvata invariata nella forma in cui si trovava nella collezione di dati iniziale:

0, 0, 0, B.

Dopo aver ritrasformato opportunamente i percorsi in istanze viene generata la versione compressa della collezione di dati iniziale, data dall'unione delle nuove istanze estratte dai percorsi e da quelle classificate in modo errato.
Quindi, la versione compressa sarà costituita dalle tre istanze mostrate di seguito:

0, 0, ?, A
0, 1, ?, B
0, 0, 0, B

Già da questo esempio si può notare un fattore di compressione pari al 40%, in quanto da cinque istanze iniziali, dopo l'applicazione della fase di compressione dell'algoritmo implementato, la versione compressa della collezione di dati sarà formata soltanto da tre istanze, due delle quali costituite dalle istanze derivate dai percorsi e la terza è l'istanza classificata in modo errato.
Dopo la fase di compressione, il dataset compresso viene utilizzato per la costruzione di un nuovo modello di classificazione. Per la generazione di questo modello viene tenuto conto del peso associato alle istanze derivate dai percorsi. Il nuovo modello di classificazione viene costruito sempre grazie all'utilizzo della tecnica dell'albero di decisione, ma è differente dal precedente e viene utilizzato per l'effettiva classificazione dei dati di test. Chiaramente i due modelli di classificazione, uno generato dalla collezione di dati iniziale e l'altro generato dalla sua versione compressa, producono differenti valori di accuratezza nella classificazione delle stesse istanze di test. L'analisi del confronto tra i risultati di accuratezza ottenuti dai due modelli verrà presa in considerazione nel Capitolo 4.
Occorre notare che in presenza di grandi moli di dati risulta difficile o a volte impossibile la creazione di un modello di classificazione utile alla compressione a partire dall'intera collezione di dati iniziale. Inoltre, anche se si riuscisse a costruire il modello, talvolta questo potrebbe risultare troppo complesso. Quindi, è possibile utilizzare soltanto una parte della collezione di dati iniziale di input per la generazione del modello utile alla compressione.

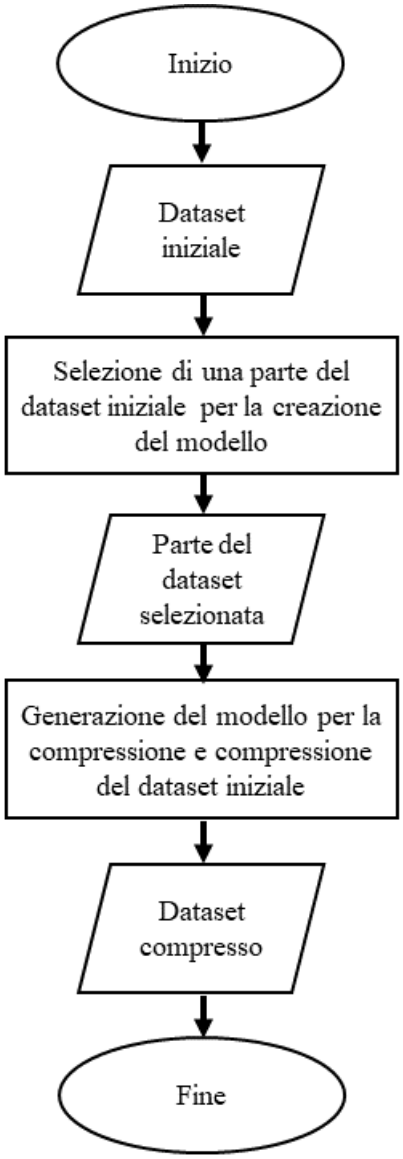


Figura 3.3 Diagrammi a blocchi generazione dataset compresso

Nella Figura 3.3 è mostrato un diagramma a blocchi che spiega al meglio il modo in cui viene generata la versione compressa della collezione di dati iniziale quando non è possibile costruire un modello di classificazione utilizzando l'intera collezione di dati. Inizialmente, viene selezionata una parte della collezione di dati di grandi dimensioni e di seguito dalla parte selezionata viene generato il modello di classificazione che verrà utilizzato per la compressione. Infine, grazie all'utilizzo di questo modello, viene effettuata la compressione dell'intera collezione di dati iniziale.

3.2 Strumenti utilizzati per la realizzazione dell'algoritmo

In questo paragrafo verranno descritti il linguaggio di programmazione Java utilizzato per l'implementazione dell'algoritmo proposto in questa tesi, Eclipse, e Weka, una collezione open-source di algoritmi di Data Mining.

3.2.1 Java

Java è un linguaggio di programmazione orientato agli oggetti che permette di sviluppare applicazioni che saranno eseguite su una Java Virtual machine (JVM) in modo da poter essere utilizzate su qualsiasi piattaforma hardware. Infatti, dopo che un programma scritto in questo linguaggio di programmazione viene compilato da un compilatore Java, viene generato un cosiddetto **byte-code**. Questo non è un linguaggio Assembly di uno specifico processore, ma un linguaggio di basso livello per una macchina virtuale, la JVM appunto. Quindi, date queste caratteristiche di compilazione, un programma scritto in Java può essere eseguito su qualsiasi piattaforma hardware. Altri componenti importanti sono il Java Runtime Environment (JRE), che contiene al suo interno la JVM e la libreria, e il Java Development Kit (JDK), costituito da tutti gli strumenti necessari per lo sviluppo di un programma Java, compreso il compilatore Java.
Il linguaggio Java ha una sintassi simile agli altri linguaggi di programmazione, ma presenta delle caratteristiche proprie di un linguaggio orientato agli oggetti. In un programma Java ogni cosa è un **oggetto**, che rappresenta un puntatore ad una zona dell'heap, una parte dello spazio di indirizzamento di un programma in esecuzione allocabile dinamicamente. In Java è definito inoltre il concetto di **classe**, che è costituita da un insieme di variabili e da **metodi**, che corrispondono a funzioni richiamabili sugli oggetti di tale classe. Sia le variabili che i metodi possono trovarsi in tre stati diversi: pubblici, privati e protetti. In un programma Java vengono quindi definite le classi che poi vengono istanziate creando così oggetti appartenenti a tali classi. Di fondamentale importanza assume il concetto di **ereditarietà** di Java nello sviluppo dell'algoritmo di questa tesi, in quanto grazie a questo concetto è possibile definire nuove classi a partire da quelle esistenti ed utilizzarne tutte le variabili e i metodi che queste presentano.

3.2.2 Eclipse

Eclipse è un IDE, integrated development environment, che consente lo sviluppo di programmi utilizzando diversi linguaggi di programmazione. Fornisce vari strumenti che favoriscono la facilità della programmazione e, inoltre, consente di caricare all'interno dei progetti dei file jar per poter importare delle librerie esterne.
Per lo sviluppo dell'algoritmo proposto in questa tesi è stato integrato nel progetto il file jar del software Weka. L'utilità di questo software verrà discussa nel Paragrafo 3.2.3.

3.2.3 Weka

Weka è una collezione di algoritmi di machine learning open-source utilizzabili per effettuare analisi di Data Mining. [4] [5] Sono presenti diversi tipi di algoritmi utilizzabili per tutte le problematiche tipiche del Data Mining, poiché consentono di effettuare preprocessing, classificazione, clustering e regressione sulle collezioni di dati. Tutti gli algoritmi disponibili presentano la possibilità di poter settare dei parametri in modo da personalizzarne l'esecuzione.
Come tecniche di preprocessing, in Weka, sono disponibili filtri classificati in supervised e unsupervised. Entrambe le tipologie sono applicabili sia ai soli attributi che costituiscono le istanze che alle istanze stesse. Esistono diverse tipologie di filtri supervised per gli attributi tra cui, ad esempio, sono presenti le classi:

- AttributeSelection: consente di selezionare solo determinati attributi tra tutti quelli delle istanze della collezione di dati sotto analisi;
- NominalToBinary: consente la conversione degli attributi di tipo nominali in binari;
- Discretize: consente di discretizzare gli attributi di tipo numerico in attributi nominali.

Se si considerano, invece, i filtri di tipo unsupervised sono presenti ulteriori tipologie di filtro come ad esempio la classe NumericToNominal, che consente di convertire attributi numerici in nominali. A differenza del filtro rappresentato dalla classe Discretize, questo prende tutti i valori numerici di un attributo, senza discretizzarli in intervalli, e li trasforma in valori nominali.
Dal punto di vista della classificazione, sono presenti diverse tecniche tra cui quelle basate sui classificatori bayesiani, sugli alberi di decisione, sulle regole di decisione e di associazione. Per quanto riguarda la tecnica di classificazione basata sugli alberi di decisione, in Weka sono presenti diversi algoritmi che consentono la costruzione del modello di classificazione, rappresentati dalle classi J48, Id3, M5P, REPTree, RandomTree e RandomForest.
Weka, inoltre, fornisce l'implementazione di algoritmi per clustering come il SimpleKMeans, che implementa l'algoritmo di clustering K-means.
Per l'utilizzo di queste tecniche, il software Weka può essere utilizzato sia attraverso l'interfaccia grafica che tramite codice all'interno di un qualsiasi programma Java. Qualsiasi sia la modalità di utilizzo che viene scelta per poter eseguire algoritmi di Data Mining di Weka, le collezioni di dati in ingresso agli algoritmi devono essere in un formato particolare, il formato **ARFF**.

3.2.3.1 Formato ARFF di una collezione di dati

Una collezione di dati per poter essere analizzata dal software Weka deve essere in formato **ARFF**, **Attribute Relationship File Format**. ARFF è un formato di rappresentazione testuale di una collezione di dati ed è costituito da due sezioni: una

sezione d'intestazione, in cui sono presenti dei **tag** seguiti dal corrispettivo valore, e una sezione relativa alle istanze della collezione di dati. Considerando la sezione iniziale del file sono presenti due tipi di tag: **@relation**, utile ad indicare il nome del dataset e deve essere posto come prima riga del file, e **@attribute**, che viene utilizzato per la descrizione degli attributi. Infatti, quest'ultimo tag è seguito dal nome dell'attributo e dal suo tipo oppure dai valori che può assumere quel determinato attributo, se di tipo nominale. Invece, la sezione relativa alle istanze è introdotta dal tag **@data**.

```
@relation prova

@attribute a1 numeric
@attribute a2 {A, B, C}
@attribute a3 {destra, sinistra}
@attribute a4 {si, no}
@attribute classe {0, 1}

@data

3.2, A, destra, si, 0
4.7, A, ,sinistra, si, 1
8, B, destra, no, 0
3.1, A, destra, si, 0
2.3, C, sinistra, si, 1
5, A, sinistra, si, 1
5.2, A, sinistra, no, 0
1.2, C, destra, no, 0
```

Figura 3.4 Esempio file in formato ARFF

La collezione di dati in formato ARFF di esempio mostrata in Figura 3.4 è denominata "prova" ed è costituita da otto istanze formate da cinque attributi: a1 di tipo numerico e i restanti di tipo nominale. Infatti, a3, a4, e classe possono assumere due valori differenti, invece, a2 può assumere tre valori: A, B e C.

3.2.3.2 Interfaccia grafica Weka

Il software Weka può essere utilizzato in due modalità differenti. In questo paragrafo verrà trattata la modalità di utilizzo tramite interfaccia grafica. In questo caso è possibile selezionare tra le diverse modalità di utilizzo presenti: Simple CLI, Explorer, Experimenter e Knowledge flow.

- Simple CLI: è una linea di comando da cui è possibile eseguire direttamente i comandi Weka;

- Explorer: consente di selezionare ed utilizzare i diversi algoritmi di Data Mining sulle collezioni di dati in input;
- Experimenter: utile per effettuare analisi statistiche;
- Knowledge flow: consente la selezione dei diversi componenti del software Weka, il posizionamento ed il collegamento tra loro per poter formare un flusso di analisi della collezione di dati in input.

3.2.3.3 Utilizzare Weka in un programma Java

Una seconda modalità di utilizzo del software Weka è l'integrazione di un file jar di Weka nel progetto relativo al programma Java che si intende implementare.
Dopo avere integrato il file jar al progetto è possibile importare le librerie per l'utilizzo di tutte le funzionalità che Weka è in grado di fornire. In questo modo è possibile implementare un'applicazione Data Mining istanziando nel codice oggetti come classificatori, filtri e tutti gli altri strumenti presenti all'interno di Weka ed utilizzare i diversi algoritmi disponibili ad essi associati.

```
DataSource source = new DataSource("file_di_prova.arff");
Instances dataset = source.getDataSet();
if (dataset.classIndex() == -1) {
    dataset.setClassIndex(dataset.numAttributes()-1);
}
```

Figura 3.5 Frammento di codice: utilizzo del software Weka in un programma Java

Nel frammento di codice in Figura 3.5 è mostrato un esempio di utilizzo del software Weka tramite codice all'interno di un programma Java. Nella libreria Weka sono presenti due classi importanti per lo sviluppo di un'applicazione Data Mining: DataSource, che consente la lettura dei file contenenti le collezioni di dati presenti in locale, e Instances, che rappresenta le istanze di una collezione di dati. In particolare, in questo codice di esempio viene letto il file in formato ARFF denominato "file_di_prova.arff", di cui ne vengono lette le istanze tramite la funzione *getDataSet()*, metodo della classe DataSource di Weka. Le istanze lette vengono conservate nella variabile denominata "dataset" di tipo Instances. Inoltre, viene settato l'ultimo attributo delle istanze presenti nella collezione di dati come etichetta di classe tramite il metodo *setClassIndex()*, richiamato sull'oggetto della classe Instances. Quest'ultimo passaggio è utile quando le istanze devono andare in ingresso ad un classificatore.
Per l'implementazione dell'algoritmo proposto in questa tesi, il software Weka è stato utilizzando in questa modalità; in particolare ne è stata anche modificata in parte la funzione di classificazione dell'algoritmo J48 della libreria Weka.

3.2.3.4 La classe J48 di Weka

J48 è una classe Java presente nella libreria Weka che implementa la versione open-source dell'algoritmo C4.5, utile per la costruzione di un modello di classificazione basato sull'albero di decisione. È possibile creare il modello ad albero settando dei parametri per la personalizzazione della sua costruzione. Tra le varie opzioni disponibili sono presenti:

- -U: permette la creazione di un albero di decisione completo, senza ridurre la dimensione tramite l'eliminazione di alcuni nodi di decisione. Questo potrebbe portare ad un problema di overfitting. Di default non è abilitato;
- -O: permette di evitare il collassare i nodi interni dell'albero di decisione in nodi foglia. Di default è abilitato;
- -C <valore di confidenza>: con questa opzione è abilitato la potatura dell'albero e viene settata una soglia di confidenza sotto la quale viene effettuato l'eliminazione del nodo. Di default ha un valore pari a 0,25;
- -M <numero di istanze>: viene settato il valore del numero minimo di istanze che devono essere presenti in un nodo foglia. Di default è settato a 2;
- -R: consente di ridurre l'errore di potatura ed è disabilitato di default;
- -N <valore>: di default ha un valore pari a 3 e consente di settare un valore corrispondente al numero di sottoinsiemi utili a ridurre l'errore di potatura;
- -B: abilita soltanto la creazione di un albero di decisione con split binari. Disabilitato di default;
- -S: questa opzione è abilitata di default e permette di sostituire una parte dell'albero di decisione con il proprio nodo figlio;
- -L: abilitato di default, consente di non cancellare il modello dell'albero di decisione dopo la sua creazione;
- -A: usa la correzione di Laplace per la generazione delle predizioni per migliorare la costruzione del modello. Disabilitato di default;
- -J: indica di non utilizzare la correzione MDL (Minimum Description Length) per lo split su attributi numerici e, di default, è disabilitato;
- -Q <seme>: serve per settare un valore random per rendere i dati casuali e di default ha valore impostato a 1.

Inoltre, la classe J48 è formata da variabili, che corrispondono ai valori delle opzioni elencate in precedenza, e da un insieme di metodi. Tra i principali metodi, utilizzati nel codice Java per l'implementazione dell'algoritmo, sono presenti:

- buildClassifier(Instances): consente la costruzione del modello di classificazione a partire dalle istanze di training passate come parametro, rappresentate dalla classe Instances presente in Weka;
- graph(): ritorna una stringa che rappresenta il modello di classificazione ad albero di decisione costruito dal metodo buildClassifier();

- classifyInstance(Instance): permette di classificare l'istanza passata come parametro ritornando come valore di tipo double che indica l'indice della classe a cui l'istanza appartiene;
- setOptions(String[]): consente di settare le opzioni elencate in precedenza. Le opzioni vengono passate a questa funzione tramite parametro come array di stringhe, in cui ogni stringa rappresenta un'opzione.

Per l'implementazione dell'algoritmo proposto in questa tesi è stata modificata la funzione della classe J48 di Weka che permette di effettuare la classificazione delle istanze. Nel Paragrafo 3.3 verrà discusso nel dettaglio la modifica apportata a tale funzione e il funzionamento dell'algoritmo realizzato.

3.3 Analisi dell'algoritmo implementato

In questo paragrafo verrà presentato in modo dettagliato l'algoritmo implementato in questa tesi. L'algoritmo di classificazione scelto, basato sulla tecnica di classificazione ad albero di decisione, è il J48 della libreria Weka. [4]

3.3.1 Struttura dell'algoritmo implementato

L'algoritmo implementato in questa tesi è strutturato in tre diverse fasi:

1. Creazione del modello di classificazione utile alla compressione utilizzando la collezione di dati etichettati iniziale o un sottoinsieme di essa
2. Fase di compressione
3. Generazione del nuovo modello di classificazione utilizzando la versione compressa della collezione di dati iniziale

Nei Paragrafi 3.3.2, 3.3.3 e 3.3.4 verranno analizzate nel dettaglio tutte e tre le fasi.

3.3.2 Generazione del modello di classificazione per la compressione

Per la costruzione del modello di classificazione utilizzato per la compressione di una collezione di dati iniziale etichettata è stata utilizzata la classe J48 di Weka, definita della libreria weka.classifiers.trees.J48. Innanzitutto, la classe J48 è stata estesa, come mostrato nel diagramma UML delle classi in Figura 3.6:

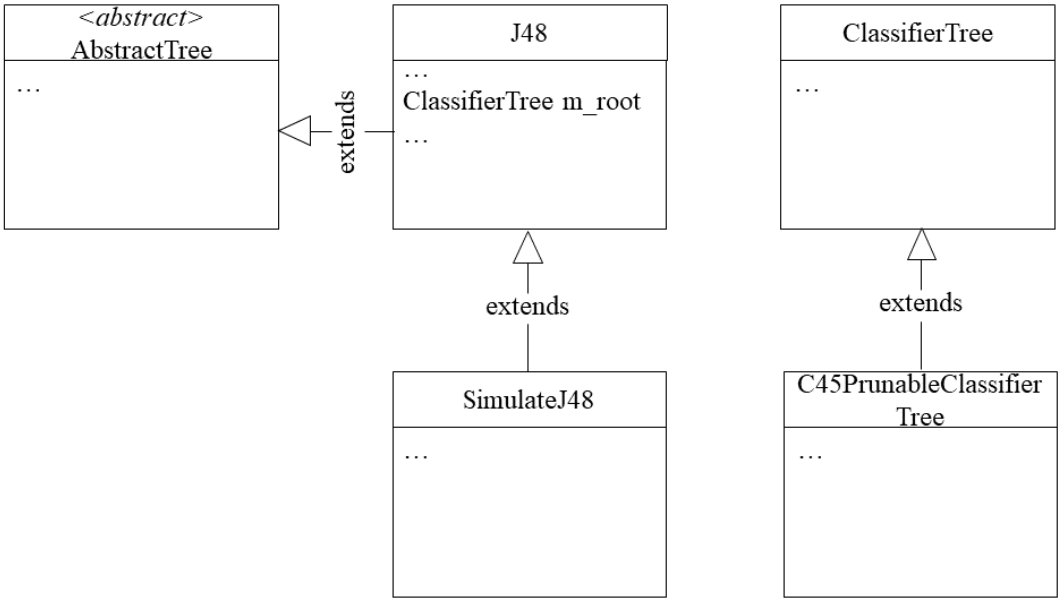


Figura 3.6 Diagramma UML delle classi

Come mostrato in Figura 3.6, è stata creata la classe SimulateJ48 che estende la classe Weka J48. Dal concetto di ereditarietà di Java è noto che da questo momento istanziando oggetti della classe SimulateJ48 è possibile utilizzare tutti i metodi della classe padre J48 e, in aggiunta, è possibile definirne di nuovi. Utilizzando la classe SimulateJ48 creata si può costruire il modello di classificazione basato sull'algoritmo J48, visto che questa estende la classe J48.

```
public class SimulateJ48 extends J48{

    public static void main(String[] args){
        DataSource source = new DataSource("dataset.arff");
        Instances training_set = source.getDataSet();
        if (training_set.classIndex() == -1) {
            training_set.setClassIndex(training_set.numAttributes()-1);
        }

        SimulateJ48 model = new SimulateJ48();
        model.buildClassifier(training_set);
        System.out.println(model.graph());
```

Figura 3.7 Frammento di codice: generazione modello di classificazione

Come si evince dal frammento di codice in Figura 3.7, dopo una fase iniziale di lettura delle istanze di training etichettate dal file in formato ARFF "dataset.arff", viene istanziato un oggetto della classe SimulateJ48, denominato "model". Successivamente,

viene richiamato il metodo *buildClassifier()* della classe SimulateJ48, ereditato dalla classe J48 di Weka, che consente la creazione del modello di classificazione ad albero di decisione utilizzando l'algoritmo J48. Questo metodo riceve come parametro le istanze di training grazie alle quali viene generato il modello iniziale che sarà utilizzato per la compressione della collezione di dati. Di default è disabilitato lo split binario nell'algoritmo J48, quindi viene creato un oggetto della classe C45PrunableClassifierTree di Weka che estende la classe ClassifierTree di Weka, che rappresenta l'albero di decisione vero e proprio. All'interno del metodo *buildClassifier()* di J48 viene chiamato il metodo *buildClassifier()* della classe ClassifierTree, che al suo interno richiama la funzione *buildTree()*, che effettivamente consente di costruire la vera struttura dell'albero di decisione. Un oggetto della classe ClassifierTree della libreria Weka, infatti, è dichiarato come variabile all'interno della classe J48 e rappresenta il puntatore alla radice dell'albero di decisione che viene costruito. Il modello di classificazione creato grazie all'utilizzo di queste classi è costituito da oggetti di tipo ClassifierTree, che ne rappresentano i nodi.
Grazie all'utilizzo del metodo *graph()*, ereditato dalla classe J48 di Weka, è possibile visualizzare, inoltre, la struttura dell'albero di decisione creato.

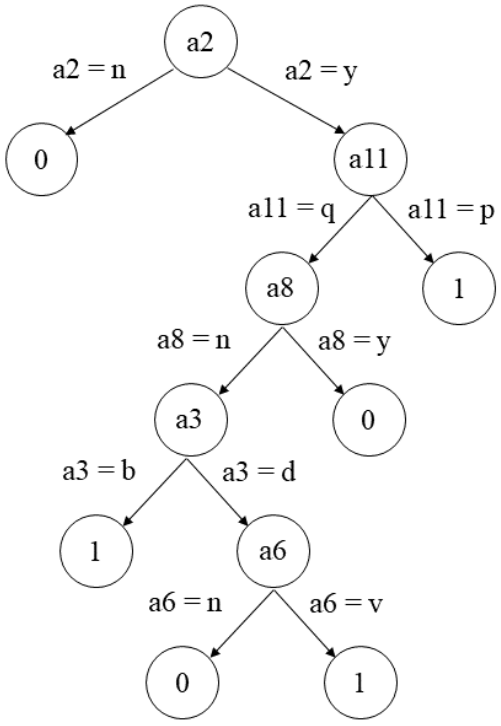


Figura 3.8 Modello di esempio generato da una collezione di dati iniziale

In Figura 3.8 è mostrato il modello di classificazione di una collezione di dati di esempio. Il modello ad albero di decisione potrebbe non presentare tutti gli attributi delle istanze

di una collezione di dati. Questo può essere dovuto al fatto che scegliendo questi particolare attributi si riesce a classificare meglio le istanze di training oppure può essere stata effettuata un'operazione di pruning, visto che di default è abilitata nell'algoritmo J48.
Il modello di classificazione verrà utilizzato per la compressione della collezione di dati iniziale, descritta nel Paragrafo 3.3.3.

3.3.3 Fase di compressione

Dopo aver creato il modello di classificazione, per poter effettuare la compressione della collezione di dati iniziale occorre utilizzare questa stessa collezione di dati etichettata come dati di test. Per la compressione è stato scelto di modificare il codice relativo alla funzione che permette di classificare un'istanza dell'algoritmo J48 di Weka. Inizialmente, è stato modificato il codice Weka della funzione *classifyInstance()* della classe J48. Questa all'interno di essa richiama la funzione *classifyInstance()* sull'oggetto della classe ClassifierTree di Weka, che rappresenta la radice dell'albero di decisione. La funzione originale di Weka svolge un calcolo della probabilità che una determinata istanza di test appartenga ad ognuna delle possibili classi utilizzando la funzione *getProbs()*. Alla fine, tra i valori di probabilità ottenuti per ognuna delle possibili classi, viene selezionato il valore massimo e viene ritornato l'indice della classe associato a quel valore di probabilità. In questo modo avviene il processo di classificazione di un'istanza di test grazie al codice della funzione standard di classificazione dell'algoritmo J48 di Weka.
Per l'implementazione della fase di compressione dell'algoritmo sono state definite due nuove funzioni: una che parte dal codice di *classifyInstance()*, chiamata *predictInstance()*, e un'altra invece che utilizza il codice di *getProbs()*, denominata *myGetProbs()*. Si parte dall'idea di differenziare le istanze secondo due tipologie: quelle classificate correttamente e quelle classificate in modo errato. I due tipi di istanze vengono salvate in due file in modo differente; infatti, le istanze erroneamente classificate vengono lasciate invariante, invece, delle istanze classificate in modo corretto viene conservato soltanto il percorso attraversato nel modello di classificazione utilizzato per la compressione ed un fattore peso. Il peso di un percorso corrispondente al numero di volte in cui quel percorso è stato attraversato dalle istanze classificate correttamente.
Nell'algoritmo proposto in questa tesi, la funzione *myGetProbs()* è richiamata all'interno della funzione *predictInstance()* e rende possibile il calcolo del valore dell'etichetta di classe da assegnare ad una determinata istanza di test ed la creazione di una stringa indicante il percorso effettuato da tale istanza. In aggiunta al codice di Weka della funzione *getProbs()*, per ogni nodo di split dell'albero di decisione attraversato ne viene salvato un valore di tipo: "nome attributo di split = valore". Inoltre, quando l'istanza raggiunge un nodo foglia, dopo aver attraversato tutto il modello di classificazione, viene salvato anche il valore della classe nella seguente modalità: "classe = valore".
La funzione *predictInstance()* viene richiamata per tutte le istanze di test, costituite da tutte le istanze della collezione di dati iniziale etichettate. Consente di calcolare e salvare in una classe creata, *PathAndPredictClass*, il percorso attraversato dall'istanza durante la

fase di classificazione, il valore dell'etichetta di classe ottenuto dalla classificazione, e anche lo stato di predizione, ossia un valore booleano che indica se la classificazione è avvenuta nel modo corretto o no, visto che le istanze di test sono etichettate. Per il calcolo di quest'ultimo valore, viene confrontato il valore iniziale dell'etichetta di classe appartenente all'istanza già etichettata e quello ottenuto dalla sua classificazione utilizzando il modello costruito nella fase precedente dell'algoritmo. Questa funzione, inoltre, conserva le indicazioni sui percorsi attraversati dalle istanze classificate correttamente ed un valore corrispondente al numero di volte in cui questi sono stati attraversati. Un esempio di un file contenente le istanze classificate correttamente, grazie all'utilizzo delle due funzioni implementate, è mostrato in Figura 3.9:

```
@relation prova

@attribute a1 {x,y}
@attribute a2 {y,n}
@attribute a3 {b,d}
@attribute a4 {a,b}
@attribute a5 {y,n}
@attribute a6 {n,v}
@attribute a7 {n,y}
@attribute a8 {n,y}
@attribute a9 {a,b}
@attribute a10 {s,d}
@attribute a11 {p,q}
@attribute class {0,1}

@data

[a2 = n, class = 0] #Traversed: 245
[a2 = y, a11 = p, class = 1] #Traversed: 135
[a2 = y, a11 = q, a8 = n, a3 = b, class = 1] #Traversed: 18
[a2 = y, a11 = q, a8 = y, class = 0] #Traversed: 5
[a2 = y, a11 = q, a8 = n, a3 = d, a6 = n, class = 0] #Traversed: 4
[a2 = y, a11 = q, a8 = n, a3 = d, a6 = v, class = 1] #Traversed: 2
```

Figura 3.9 Esempio di file contenente i percorsi

Dalla Figura 3.9 si può notare che sono state correttamente classificate 409 istanze di una collezione di dati di esempio e, dato che sono stati attraversati soltanto sei percorsi, si ha un file con un numero di righe pari a sei. È possibile calcolare il numero di istanze classificate in modo corretto dalla somma di tutti i pesi associati a tutti i percorsi attraversati. Importante è l'indicazione sul numero di volte in cui il percorso è stato attraversato e, nel Paragrafo 3.3.4, verrà spiegato il modo in cui verrà utilizzato. Occorre notare, quindi, che per la successiva costruzione della versione compressa della collezione di dati iniziale di queste 409 istanze correttamente classificate ne verranno solamente considerate soltanto sei. Le nuove istanze verranno estratte da questi percorsi ottenuti, come mostrato nel Paragrafo 3.3.3.1. Questa è una prima indicazione sul fattore di compressione che viene applicato alla collezione di dati iniziale.

Una considerazione importante è che in questa fase di compressione le istanze di test che presentano un missing value per un attributo di split del modello di classificazione non vengono tenute in considerazione durante la creazione dei due file e vengono scartate.
Infine, la fase di compressione viene completata con la generazione della versione compressa della collezione di dati il cui procedimento è descritto nel Paragrafo 3.3.3.1.

3.3.3.1 Generazione della versione compressa della collezione di dati

Dopo la generazione dei due file contenenti uno le istanze classificate in modo corretto nel formato mostrato in Figura 3.9 e l'altro contenente le istanze classificate erroneamente nel formato in cui si trovavano nella collezione di dati iniziale, si passa alla fase di generazione della versione compressa della collezione di dati. Occorre considerate in un primo momento il file contenente i percorsi attraversati nel modello dalle istanze durante la loro classificazione. Per la creazione della versione compressa vengono analizzate tutte le righe contenute nel file rappresentanti i percorsi. Le nuove istanze vengono create nel formato iniziale utilizzando soltanto i valori degli attributi presenti nei percorsi. Gli attributi delle istanze che non presentano alcun valore nel percorso vengono sostituite con il simbolo '?' indicante un missing value per quell'attributo. Un esempio di come avviene questo procedimento è mostrato di seguito:
In riferimento al file mostrato in Figura 3.9, i percorsi attraversati dalle istanze sono i seguenti:

1. a2 = n, class = 0
2. a2 = y, a11 = p, class = 1
3. a2 = y, a11 = q, a8 = n, a3 = b, class = 1
4. a2 = y, a11 = q, a8 = y, class = 0
5. a2 = y, a11 = q, a8 = n, a3 = d, a6 = n, class = 0
6. a2 = y, a11 = q, a8 = n, a3 = d, a6 = v, class = 1

Considerando quindi questi sei percorsi e le modalità di creazione delle istanze descritte, verranno create le seguenti istanze:

1. ?, n, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, 0
2. ?, y, ?, ?, ?, ?, ?, ?, ?, ?, p, 1
3. ?, y, b, ?, ?, ?, ?, n, ?, ?, q, 1
4. ?, y, ?, ?, ?, ?, ?, y, ?, ?, q, 0
5. ?, y, d, ?, ?, n, ?, n, ?, ?, q, 0
6. ?, y, d, ?, ?, v, ?, n, ?, ?, q, 1

Come si può notare dalle nuove istanze generate, sono presenti diversi valori mancanti per gli attributi che non costituiscono i nodi di split nel modello di classificazione ad albero di decisione utilizzato per la compressione.
Di queste istanze, per la successiva generazione del modello di classificazione, viene tenuto conto del peso associato al percorso da cui sono state costruite. Infatti, ad ogni percorso è associato un peso, che corrisponde al numero di volte in cui è stato attraversato

durante la classificazione dalle istanze correttamente classificate. Infine, le istanze così generate vengono adesso salvate in un file in formato ARFF.
La nuova collezione di dati, che rappresenta la versione compressa di quella iniziale, non è formata soltanto da queste istanze, ma in aggiunta si considerano le istanze classificate in modo errato. Quest'ultime vengono lette dal file in cui erano state salvate in precedenza e salvate anch'esse nel file ARFF. In questo modo il file ARFF generato conterrà la versione compressa della collezione di dati originale utilizzando l'algoritmo implementato in questa tesi.
Utilizzando la nuova collezione di dati è possibile generare un nuovo modello che verrà utilizzato per classificare le vere istanze di test, come spiegato nel Paragrafo 3.3.4.

3.3.4 Generazione del modello di classificazione a partire dal dataset compresso

Dopo la generazione del file contenente la versione compressa della collezione di dati iniziale, le istanze presenti in questo file vengono utilizzate come dati di training per la costruzione del modello che verrà utilizzato per l'effettiva classificazione delle istanze di test. Per ogni istanza viene considerato il peso associato ad essa, poiché adesso il numero di istanze è minore, grazie alla fase di compressione effettuata, ma ogni istanza che deriva da un percorso attraversato dell'albero di decisione ha un peso differente nella costruzione del modello di classificazione. Il peso di ogni istanza di training viene tenuto conto grazie all'utilizzo di un metodo *setWeigth(valore)* della classe Instance di Weka.

```
DataSource source = new DataSource("dataset_compresso.arff");
Instances training_set = source.getDataSet();
if (training_set.classIndex() == -1) {
    training_set.setClassIndex(training_set.numAttributes()-1);
}

training_set.get(0).setWeight(245);
```

Figura 3.10 Frammento di codice: settare il peso ad un'istanza di training

Dal frammento di codice di esempio in Figura 3.10, dopo la lettura delle istanze dalla collezione di dati dal file denominato "dataset_compresso.arff", viene settato un fattore peso pari a 245 alla prima istanza della collezione di dati. Per la costruzione del modello di classificazione, quindi, verrà tenuto conto di questo fattore di peso presente nella prima istanza di training. Il nuovo modello di classificazione generato sarà differente rispetto a quello utilizzato per la compressione della collezione di dati. Per l'effettiva classificazione di dati di test non etichettati verrà utilizzato il modello generato grazie all'utilizzo della versione compressa della collezione di dati iniziale. Per stimare la qualità dell'algoritmo implementato in questa tesi, di fondamentale importanza sono i

valori di compressione e di accuratezza della classificazione ottenuti dal modello così costruito. Questi valori verranno presentati ed analizzati nel Capitolo 4 dedicato alla parte sperimentale di questa tesi.

Capitolo 4
Sezione sperimentale

In questo capitolo verranno analizzati tutti i risultati degli esperimenti svolti in modo tale da mostrare il livello di qualità dell'algoritmo implementato. Verrà analizzato sia il grado di compressione che la qualità del modello di classificazione generato dalla versione compressa di una collezione di dati tramite l'utilizzo dell'algoritmo proposto in questa tesi. Nella prima parte del capitolo, dopo una definizione del valore di accuratezza, parametro di qualità utilizzato per la valutazione dei risultati degli esperimenti, verranno analizzate due delle principali metodologie di esperimenti effettuati. Successivamente, verranno presentati i dataset utilizzati e mostrati i risultati dei test effettuati su di essi con i relativi commenti.

4.1 Valutazione di un modello di classificazione

La valutazione della qualità di un modello di classificazione generato da una tecnica di classificazione è di fondamentale importanza. È corretto, quindi, utilizzare delle metriche di valutazione come l'accuratezza, definita nel Paragrafo 4.1.1, e anche delle metodologie note che spiegano il modo in cui affrontare gli esperimenti utili a fornire delle indicazioni riguardo la qualità di un modello di classificazione. Tra le metodologie da utilizzare per questa stima di qualità troviamo: **Cross validation** e **Holdout**. Queste due differenti tipologie di approccio partono entrambe dall'idea di partizionare una collezione di dati in training set e test set, ma in maniera differente, per poterne effettuare dei test. Visto la loro utilità nella valutazione della qualità di un modello di classificazione, nei Paragrafo 4.1.2 e 4.1.3 verranno analizzate entrambe.

4.1.1 Accuratezza

L'accuratezza è uno dei principali parametri utilizzati per la valutazione della qualità di classificazione di un modello di classificazione. [1] Questo valore, però, non è sempre affidabile, poiché non è adatto alla stima della qualità di un classificatore in presenza di una collezione di dati costituita da una distribuzione non omogenea delle etichette di classe. Considerando le istanze di una collezione di dati, l'accuratezza viene misurata utilizzando la seguente relazione:

$$Accuratezza = \frac{\#istanze\ correttamente\ classificate}{\#istanze\ totali}$$

4.1.2 Cross Validation

La Cross Validation è una tipologia di approccio utile alla valutazione delle prestazioni di un modello di classificazione generato da una determinata tecnica di classificazione. Questa è una delle tecniche maggiormente utilizzate, ma non è indicata per la stima del livello di accuratezza di una collezione di dati costituita da un'elevata cardinalità. Come la tecnica di Holdout, presentata nel Paragrafo 4.1.3, parte dall'idea di partizionare la collezione di dati in sottoinsiemi, ma a differenza di questa non effettua una partizione fissa in due sottoinsiemi. Infatti, utilizzando questa metodologia di approccio, dalla collezione di dati vengono generati *k* sottoinsiemi diversi. Tra i *k* sottoinsiemi, il *k*-esimo è considerato come test set e invece i restanti *k-1* sottoinsiemi vengono utilizzati come training set per la generazione del modello di classificazione. Successivamente, questo approccio viene applicato in maniera ricorsiva per tutti i *k* sottoinsiemi. Alla fine tutti i *k* sottoinsiemi verranno utilizzati per la il test del modello di classificazione.
Un caso particolare della Cross Validation è chiamato **leave-one-out**, in cui il valore di *k* è uguale a 1. Quindi, si hanno un numero di sottoinsiemi pari alla dimensione della collezione di dati sotto analisi. Occorre, però, utilizzare questo caso particolare per collezioni di dati costituite da bassa cardinalità, in quanto la valutazione dell'accuratezza impiegherebbe una quantità di tempo elevata a causa dei numerosi test da effettuare.
Nel Paragrafo 4.5 verranno mostrati i risultati dei test svolti tramite la Cross Validation con un valore del parametro *k* pari a 10.

4.1.3 Holdout

Holdout è una metodologia di valutazione della qualità di un modello di classificazione che parte dall'idea di partizionare la collezione di dati da analizzare in due sottoinsiemi ed è efficiente per la stima di collezioni di dati di grandi dimensioni. I due sottoinsiemi vengono creati in maniera fissa e sono così costituiti:

- $\frac{2}{3}$ della collezione di dati iniziale viene utilizzato come training set
- $\frac{1}{3}$ rimanente viene utilizzato come test set

Per ottenere risultati più veritieri riguardo la valutazione della qualità di un modello di classificazione occorre ripetere il test diverse volte per poi fare una media dei valori di accuratezza ottenuti. Per i test svolti ed analizzati nel Paragrafo 4.6, la tecnica di Holdout è stata ripetuta cinque volte e successivamente è stata fatta una media tra i cinque diversi risultati ottenuti.

4.2 Dataset utilizzati

I dataset utilizzati per i test svolti in questa tesi sono quelli dell'UCI Machine Learning Repository. [6] Sono stati scelti 31 dataset differenti in modo da valutare al meglio le prestazioni dell'algoritmo implementato, dato che questi sono costituiti da differenti

numeri e tipologie di attributi. Inoltre, i dataset scelti presentano anche differenti valori di cardinalità ed un attributo di classe di tipo nominale, così da poter essere analizzati dalla tecnica di classificazione ad albero di decisione basata sull'algoritmo J48 di Weka.

Nome del dataset	Numero di istanze	Numero di attributi
adultd	30162	15
austrad	690	15
autod	205	26
breastd	699	11
cleved	303	14
diabetesd	768	9
germand	1000	21
glassd	214	10
heartd	270	14
hepatid	155	20
horsed	368	23
ionod	351	35
irisd	150	5
letter-recognitiond	20000	17
mushroom	8124	23
pendigitsd	10992	17
pimad	768	9
satd	6435	37
segmentationd	2310	20
sickd	2800	30
sonard	208	61
soybean-large	307	36
soybean-s	47	36
splice	3175	61
tic-tac	958	10
vehicled	846	19
voting	435	17
waveformd	5000	22
wined	178	14
yeastd	1484	9
zood	101	17

Tabella 4.1: Dataset utilizzati

Nella prima colonna della Tabella 4.1 sono elencati i nomi di tutti i dataset utilizzati per gli esperimenti. Nelle altre due colonne, invece, sono mostrati il numero sia delle istanze

che degli attributi che costituiscono i dataset utilizzati. Come si può notare sono presenti collezioni di dati di diversa dimensione; questo è utile per poter testare la qualità dell'algoritmo implementato in questa tesi in diverse condizioni e per diverse tipologie di dataset.
Occorre far presente che alcuni tra i dataset elencati nella Tabella 4.1 sono stati sottoposti ad una fase di preprocessing per discretizzare gli attributi numerici, fase fondamentale per poter utilizzare l'algoritmo J48 di Weka. Inoltre, questi dataset sono stati convertiti in formato ARFF grazie all'utilizzo del software Weka, per rendere possibile l'applicazione degli algoritmi di classificazione presenti in Weka.

4.3 Valori di compressione

In questo paragrafo verrà mostrato uno dei principali risultati degli esperimenti effettuati sui dataset elencati nel Paragrafo 4.2, ottenuto grazie all'utilizzo dell'algoritmo implementato in questa tesi. L'algoritmo proposto, come ampiamente spiegato nel Capitolo 3, ha come obiettivo la compressione di dati etichettati finalizzato alla classificazione di grandi moli di dati. Quindi, a partire da una collezione di dati iniziale, è in grado di generarne una versione compressa e successivamente generare un modello di classificazione utilizzando come dati di training il dataset compresso. Questo nuovo modello così generato deve consentire la classificazione di istanze di test senza un'eccessiva perdita di accuratezza rispetto al modello di classificazione costruito utilizzando come dati di training la collezione di dati iniziale di grandi dimensioni. Dato che l'algoritmo ha come primo obiettivo la compressione di grandi moli di dati occorre valutarne la percentuale di compressione che si riesce ad ottenere in condizioni diverse, quindi è utile testarlo su collezioni di dati di diverse dimensioni e tipologie. Nella Tabella 4.2 sono mostrati i valori di compressione ottenuti applicando l'algoritmo implementato in questa tesi su tutti i 31 dataset, presi dall'UCI Machine Learning Repository, elencati nella Tabella 4.1.

Nome del dataset	n° istanze	n° percorsi	n° errori	%compressione
adultd	30162	248	3878	86,32
austrad	690	10	79	87,10
autod	205	38	14	74,63
breastd	699	16	17	95,28
cleved	303	20	34	82,18
diabetesd	768	13	154	78,26
germand	1000	64	176	76,00
glassd	214	21	38	72,43
heartd	270	8	34	84,44
hepatid	155	6	18	84,52
horsed	368	9	48	84,51

ionod	351	16	16	90,88
irisd	150	3	6	94,00
letter-recognitiond	20000	3304	1519	75,89
mushroom	8124	19	0	99,77
pendigitsd	10992	1146	508	84,95
pimad	768	13	154	78,26
satd	6435	495	463	85,11
segmentationd	2310	139	52	91,73
sickd	2800	15	45	97,86
sonard	208	16	16	84,62
soybean-large	307	34	14	84,36
soybean-s	47	4	0	91,49
splice	3175	123	117	92,44
tic-tac	958	90	60	84,34
vehicled	846	84	127	75,06
voting	435	6	9	96,55
waveformd	5000	385	741	77,48
wined	178	14	0	92,13
yeastd	1484	59	544	59,37
zood	101	9	1	90,10

Tabella 4.2: Valori di compressione in percentuale ottenuti

La Tabella 4.2 elenca nelle prime due colonne le caratteristiche dei dataset e, nelle tre successive, i risultati ottenuti dalla fase di compressione dell'algoritmo generando il modello per la compressione tramite l'intera collezione di dati in ingresso e applicandolo all'intera collezione stessa. Nella fase di compressione le istanze vengono suddivise in due tipologie e salvate in due modi differenti: di quelle classificate in modo corretto ne vengono salvati i percorsi attraversati nel modello di classificazione mentre quelle erroneamente classificate vengono lasciate invariate. Nella terza colonna, denominata "Numero di percorsi" sono mostrati il numero di percorsi generati dalla fase di compressione per tutti e 31 i dataset. Questi valori non indicano il numero di istanze correttamente classificate dall'algoritmo in quanto ad ogni percorso è associato un fattore peso che ne indica la frequenza di attraversamento da parte delle istanze classificate in modo corretto. Quindi, per il calcolo del numero di istanze classificate correttamente occorre sommare tutti i pesi di tutti i percorsi ottenuti dalla compressione. Invece, nella quarta colonna, sono elencati il numero di istanze classificate in modo errato dal modello di classificazione utilizzato per la compressione. Grazie a questi due valori è possibile quindi stimare la percentuale di compressione ottenuta, dal momento che la versione compressa di un dataset sarà costituita da un numero di istanze pari alla somma di questi due valori ottenuti. Quindi, nell'ultima colonna, per ogni dataset analizzato, è mostrata la percentuale di compressione ottenuta dall'applicazione dell'algoritmo implementato.

Considerando tutti i valori elencati in questa colonna, la percentuale di compressione ottenuta è elevata per tutti i dataset presi in considerazione. Nella maggior parte dei casi è di molto superiore all'80% e solo per un dataset il risultato di compressione risulta inferiore al 70%, il dataset denominato "yeastd". Grazie all'analisi di questi risultati ottenuti, è evidente che l'approccio utilizzato dall'algoritmo implementato in questa tesi genera valori di compressione veramente elevati, poiché, nei dataset analizzati, in media si ottiene una percentuale di compressione pari al 85%. Quindi, dal punto di vista della compressione è possibile affermare che l'algoritmo è di buona qualità. Visto che l'obiettivo dell'algoritmo non è solo la compressione di grandi moli di dati ma anche la generazione di un modello di classificazione a partire dalla loro versione compressa che consenta di etichettare i dati mantenendo una certa qualità, risulta necessario anche la valutazione di un'altra metrica di fondamentale importanza, l'accuratezza. Occorre valutare l'accuratezza del modello di classificazione generato a partire dalla versione compressa di una collezione di dati e paragonarla ai valori ottenuti utilizzando un modello di classificazione costruito a partire dalla versione iniziale di grandi dimensioni. Dall'analisi di questi ulteriori risultati sarà possibile valutare la qualità complessiva dell'algoritmo implementato in questa di tesi.

4.4 Accuratezza del modello di classificazione generato dal dataset iniziale e dalla sua versione compressa

Come espresso nel Paragrafo 4.3, oltre agli importanti valori di compressione, per un'analisi completa dell'algoritmo implementato occorre valutare e confrontare l'accuratezza del modello di classificazione generato sia a partire dalla versione compressa di una collezione di dati che utilizzando la sua versione originale di grandi dimensioni. Di seguito verranno mostrati i valori di accuratezza ottenuti utilizzando come dati di test l'intera collezione di iniziale e come training set per la generazione del modello di classificazione verrà utilizzato inizialmente l'intera collezione di dati iniziale ed in seguito la sua versione compressa. La tecnica di classificazione utilizzata per i test è quella basata sull'albero di decisione costruito tramite l'algoritmo J48 di Weka.

Nome del dataset	**%accuratezza**	
	(1)	**(2)**
adultd	87,14	70,87
austrad	88,55	87,25
autod	93,17	89,76
breastd	97,42	97,42
cleved	88,45	88,45
diabetesd	79,95	77,21
germand	82,4	77,3
glassd	82,24	82,24

heartd	87,41	87,41
hepatid	87,74	87,1
horsed	86,96	86,96
ionod	95,44	95,44
irisd	96	96
letter-recognitiond	92,41	92,21
mushroom	100	99,8
pendigitsd	95,38	94,96
pimad	79,95	77,21
satd	92,81	90,94
segmentationd	97,75	97,66
sickd	98,32	98,14
sonard	92,31	92,31
soybean-large	95,11	89,58
soybean-s	100	36,17
splice	96,32	96,13
tic-tac	93,74	83,61
vehicled	84,99	82,51
voting	97,24	97,01
waveformd	85,18	82,12
wined	100	100
yeastd	63,34	63,21
zood	99,01	97,03

Tabella 4.3: Confronto tra i valori di accuratezza

Nella Tabella 4.3 sono mostrati le percentuali di accuratezza ottenute dai test effettuati su tutti e 31 i dataset. I valori di accuratezza associati alla colonna denominata "%accuratezza (1)" in Tabella 4.3 sono stati ricavati utilizzando la collezione di dati iniziale sia come training set, per la generazione del modello di classificazione, sia come test set. Invece, i valori elencati in "%accuratezza (2)" sono stati ottenuti generando il modello di classificazione tramite la versione compressa ed effettuando il test con le istanze della collezione di dati iniziale. Dalla Tabella 4.3 è evidente che i valori associati a quest'ultima colonna sono quasi sempre inferiori rispetto a quelli elencati nella colonna "%accuratezza (2)" tranne per alcuni casi in cui i due valori sono uguali. Occorre notare, però, che i valori differiscono soltanto per un esiguo valore di percentuale. Questo sta ad indicare che anche raggiungendo un livello di compressione abbastanza elevato grazie all'algoritmo implementato in questa tesi, come dimostrano i risultati della Tabella 4.2, si riescono a raggiungere pressoché gli stessi valori di accuratezza sia che si utilizzi la collezione di dati iniziale come training set sia la sua versione compressa.
Dai dati in Tabella 4.3, occorre mettere in risalto una riduzione eccessiva della percentuale di accuratezza relativa all'analisi effettuata sul dataset denominato "soybean-

s". Infatti, questo è un caso particolare, in quanto dalla fase di compressione dell'algoritmo viene generata una versione troppo approssimativa della collezione di dati iniziale. La versione compressa di questo dataset è costituita soltanto da quattro istanze con il relativo peso, che risulta essere troppo generica per la costruzione di un modello di classificazione accurato per un dataset di questo tipo.
Dai risultati ottenuti, elencati nella Tabella 4.3, è chiaro che la compressione applicata ai dataset considerati porta ad una differenza minima tra i due valori di accuratezza. Un confronto tra i valori di compressione e la differenza tra i due valori di accuratezza ottenuta dai test è mostrato nei Grafici 4.1 e 4.2:

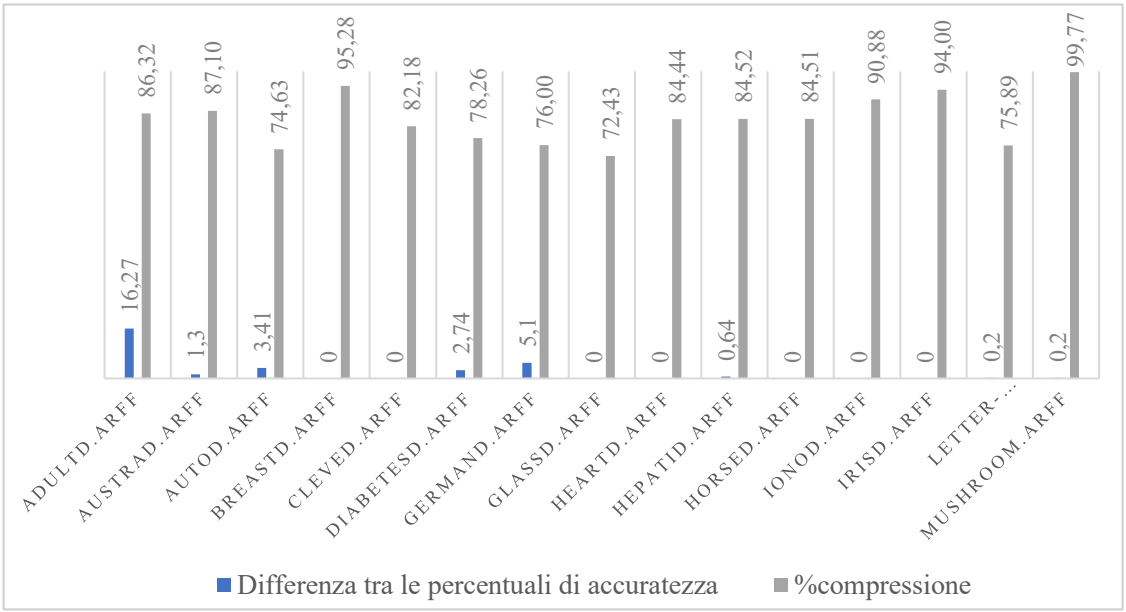


Grafico 4.1: Confronto valori di compressione e differenza di accuratezza tra i due modelli – parte I

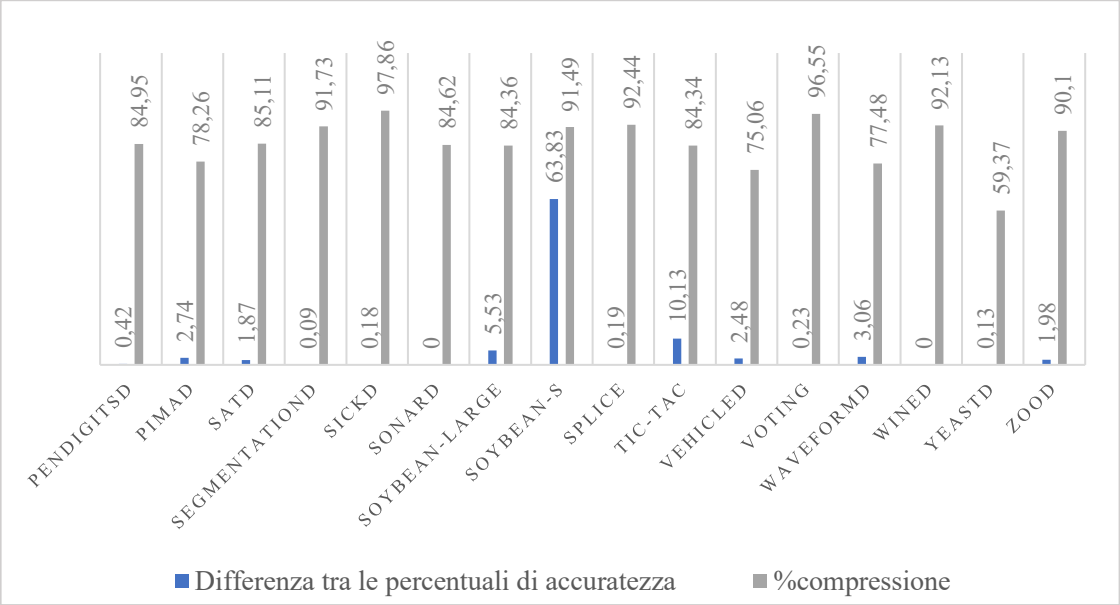


Grafico 4.2: Confronto valori di compressione e differenza di accuratezza tra i due modello – parte II

Dai Grafici 4.1 e 4.2 è evidente che i valori di compressione risultano veramente elevati rispetto alla differenza tra i valori di accuratezza ottenuti addestrando un modello di classificazione tramite la collezione di dati iniziale e la sua versione compressa generata grazie all'applicazione della fase di compressione dell'algoritmo proposto in questa tesi. Osservando la percentuale indicante la differenza tra i valori di accuratezza è presente un solo valore elevato, quello dovuto alla bassa accuratezza ottenuta dal modello di classificazione generato dalla versione compressa del dataset denominato "soybean-s". Per tutti gli altri dataset analizzati, invece, i valori sono quasi sempre intorno allo 0%. È chiaro che, dalla valutazione di questi test, la qualità complessiva dell'algoritmo è buona, ma occorre effettuare ulteriori test per poter fornire delle prove sempre più concrete del suo livello di qualità.

4.4.1 Accuratezza dopo la sostituzione dei missing value nel dataset compresso

Per generare la versione compressa della collezione di dati inziale si tiene conto sia delle istanze estratte dai percorsi attraversati dalle istanze correttamente classificate che di quelle classificate in modo errato. Le istanze estratte dai percorsi vengono generate solo dai valori degli attributi presenti in quei percorsi. Ne consegue la creazione di istanze contenenti missing value, poiché i percorsi non sono costituiti da tutti gli attributi che un'istanza presenta nella collezione di dati iniziale, ma soltanto dagli attributi di split attraversati nel modello di classificazione utilizzato per la compressione. Per i test

mostrati nel Paragrafo 4.4, il dataset compresso è costituito dalle istanze create dai percorsi con la presenza dei missing value. È possibile sostituire i missing value delle istanze in qualche modo e costruire così un nuovo dataset compresso. Quindi, è utile effettuare un confronto tra i valori di accuratezza ottenuti dal modello di classificazione generato dalle due diverse versioni del dataset compresso.
Nei risultati in Tabella 4.4 dei test svolti, i missing value di tutte le istanze del dataset compresso sono stati sostituiti con il valor medio degli attributi che presentano un valore mancante.

Nome del dataset	**%accuratezza**	
	(1)	**(2)**
adultd	70,87	22,8
austrad	87,25	28,2
autod	89,76	52,4
breastd	97,42	56,4
cleved	88,45	52
diabetesd	77,21	38
germand	77,3	39
glassd	82,24	56,2
heartd	87,41	39,6
hepatid	87,1	40
horsed	86,96	37,4
ionod	95,44	60,2
irisd	96	83,2
letter-recognitiond	92,21	50,2
mushroom	99,8	97
pendigitsd	94,96	48,4
pimad	77,21	43
satd	90,94	39,8
segmentationd	97,66	85
sickd	98,14	28,6
sonard	92,31	66,4
soybean-large	89,58	61,6
soybean-s	36,17	85,6
splice	96,13	59,4
tic-tac	83,61	54,4
vehicled	82,51	44,6
voting	97,01	72,2
waveformd	82,12	43,2
wined	100	92,6
yeastd	63,21	40,6

zood	97,03	92,2

Tabella 4.4: Percentuali di accuratezza tra due diverse versioni del dataset compresso

La Tabella 4.4 mostra, per tutti e i 31 dataset, i due diverse percentuali di accuratezza ottenute utilizzando come dati di training per la generazione del modello di classificazione le due diverse versioni del dataset compresso. Per i risultati elencati nella colonna "%accuratezza (1)" è stato utilizzato il dataset compresso senza la sostituzione dei missing value per l'apprendimento del classificatore. Per quelli mostrati nella colonna "%accuratezza (2)", invece, il modello è stato costruito considerando come dati di training il dataset compresso in cui sono stati sostituiti i valori mancanti con il valor medio per quegli attributi. Dal confronto tra le percentuali di accuratezza ottenute per tutti i 31 dataset nei due diversi test è evidente la presenza di una perdita di accuratezza utilizzando come dati di training il dataset compresso ottenuto dalla sostituzione dei valori mancanti con il valor medio. In tutti i dataset analizzati, tranne per quello denominato "soybean-s", le percentuali di accuratezza in Tabella 4.4 della colonna (2) sono inferiori rispetto a quelle elencate nella (1). In media, utilizzando il dataset compresso senza missing value, si ha una perdita di accuratezza pari al 32%.

4.5 Risultati della Cross Validation

La tecnica della Cross Validation, come spiegato nel Paragrafo 4.1.2, è ampiamente utilizzata per la valutazione della qualità di un modello di classificazione. Per effettuare i test tramite Cross validation sono state utilizzate differenti tecniche di classificazione della libreria di Weka: quella basata sull'albero di decisione con algoritmo J48 e quella basata sulle regole di classificazione con due diversi algoritmi di costruzione delle regole, Naive Bayes e JRip. Per i test effettuati tramite le diverse tecniche di classificazione è stato utilizzato un valore del parametro *k* della Cross validation pari a 10. Nella Tabella 4.5 sono mostrati i risultati della Cross Validation utilizzando in un primo momento la collezione di dati iniziale e, successivamente, la sua versione compressa, per tutte e tre le tecniche di classificazione. Occorre far presente che per i valori ottenuti la collezione di dati iniziale è stata lasciata invariata, invece, nella sua versione compressa le istanze formate dai percorsi sono state ripetute in accordo al fattore di peso del percorso da cui l'istanza era stata creata. Per esempio se un percorso fosse attraversato 15 volte, allora l'istanza generata da questo sarebbe ripetuta 15 volte nella versione compressa della collezione di dati iniziale. Solo in questi test il dataset compresso è stato modificato secondo questa modalità poiché, tramite Cross validation, non è stato possibile settare un fattore peso alle istanze estratte dai percorsi come spiegato al Paragrafo 3.3.4. In definita, adesso il dataset "compresso" avrà la stessa cardinalità del dataset iniziale, a meno delle istanze che presentavano missing value negli attributi di split del modello di classificazione, ma sarà in una forma diversa. Infatti, sarà costituito dalla ripetizione delle istanze costruite dai percorsi e da quelle erroneamente classificate.

Nome del dataset	J48		Naive Bayes		JRip	
	(1)	(2)	(1)	(2)	(1)	(2)
adultd	86,20	95,83	83,60	92,87	83,22	95,81
austrad	86,52	85,94	85,22	90,29	87,25	95,07
autod	80,49	82,18	67,80	85,64	77,07	78,71
breastd	95,99	96,70	97,14	96,13	94,85	97,42
cleved	78,55	84,51	83,83	85,19	80,53	90,57
diabetesd	77,99	76,78	78,26	79,95	77,21	91,80
germand	72,90	73,70	75,40	78,50	70,90	91,20
glassd	76,64	78,50	73,36	74,30	72,43	78,97
heartd	83,70	86,30	84,07	86,30	84,44	94,44
hepatid	81,94	85,33	85,16	89,33	78,71	96
horsed	84,51	86,14	79,35	86,14	84,51	95,65
ionod	89,74	93,45	90,60	98,29	92,59	97,44
irisd	93,33	96	94,67	96	94	96
letter-recognitiond	78,85	81,68	73,94	69,15	76,30	74,26
mushroom	100	100	95,83	100	100	100
pendigitsd	88,82	90	87,87	89,31	92,07	89,66
pimad	77,47	77,99	77,86	80,21	76,69	90,89
satd	84,40	87,23	82,42	87,41	84,10	88,95
segmentationd	94,81	95,84	91,34	94,94	93,16	95,54
sickd	97,93	97,56	97,18	97,56	97,68	98,60
sonard	79,81	87,98	85,58	94,23	79,81	96,15
soybean-large	86,39	88,62	90,55	91,03	85,99	93,10
soybean-s	97,87	100	100	100	97,87	100
splice	93,76	95,02	95,65	97,01	94,33	95,62
tic-tac	83,61	82,36	70,04	86,01	97,39	94,68
vehicled	72,34	77,54	62,88	73,40	69,39	84,16
voting	96,32	96,40	90,11	99,52	95,40	98,56
waveformd	75,36	78,02	81,76	86,30	76,08	88,20
wined	93,26	96,07	98,88	97,75	96,07	98,88
yeastd	59,30	60,44	59,10	61,79	59,10	70,01
zood	97,08	95,05	93,07	99,01	88,12	97,03

Tabella 4.5: Confronto accuratezza tramite Cross Validation

Le colonne “J48 (1)” e “J48 (2)”in Tabella 4.5 mostrano i valori di accuratezza della Cross Validation ottenuti applicando l’algoritmo J48. Considerando la colonna “J48 (1)”, i valori di accuratezza sono ottenuti applicando la Cross Validation sulla collezione di dati iniziale, invece, in quella denominata “J48 (2)” sono elencati i risultati conseguiti utilizzando il dataset “compresso”. Quest’ultimo porta a risultati di accuratezza superiori nella maggior parte dei dataset analizzati rispetto a quelli ottenuti grazie all’utilizzo della

collezione di dati iniziale. Per quanto riguarda, invece, le colonna “Naive Bayse (1)” e “Naive Bayse (2)” in Tabella 4.5, mostrano i valori della Cross Validation ottenuti utilizzando la tecnica di classificazione basata su regole Naive Bayes applicata rispettivamente al dataset iniziale e alla sua versione “compressa”. I risultati sono paragonabili a quelli ottenuti dalla tecnica di classificazione basata sull’albero di decisione con algoritmo J48. Infine, le colonne “JRip (1)” e “JRip (2)” elencano i valori di accuratezza relativi all’applicazione dalla Cross Validation utilizzando l’algoritmo JRip, altra tecnica di classificazione basata su regole. Anche questi risultati seguono la tendenza delle analisi effettuate con le altre due tecniche di classificazione.

4.6 Valori di accuratezza tramite Holdout

Ulteriori test sono stati effettuati su tutti e 31 i dataset utilizzando la metodologia denominata Holdout. Inizialmente, la collezione di dati iniziale è stata divisa in due parti ed in maniera random il 66% delle istanze sono state considerate come training set e le restanti come test set. Il primo test è stato effettuato generando il modello di classificazione tramite questo training set e poi, per valutarne l’accuratezza, è stato testato tramite il test set, costituito dal 33% delle istanze del dataset iniziale. Successivamente, un ulteriore test è stato svolto generando una versione compressa delle istanze di training iniziali, tramite la fase di compressione dell’algoritmo proposto in questa tesi. Per la generazione della versione compressa è stato utilizzato un modello di classificazione costruito tramite il training set costituito dal 66% delle istanze della collezione di dati iniziali. La percentuale di compressione ottenuta per le istanze di training di tutti e 31 i dataset utilizzati è elencata nella colonna “%compressione” della Tabella 4.6. La maggior parte dei valori di compressione ottenuti sono superiori al 80% ed in alcuni dataset si raggiunge un percentuale oltre il 90%. Di contro, per il dataset denominato “yeastd”, la percentuale di compressione è inferiore al 60%. In media si è raggiunta un percentuale di compressione pari al 84%.
Dopo la generazione di queste nuove istanze di training, che costituiscono una versione compressa delle precedenti, sono state utilizzate per la costruzione del modello di classificazione. Per testare l’accuratezza della classificazione del modello così generato è stato utilizzato lo stesso test set precedente ed i risultati ottenuti sono mostrati nella Tabella 4.6.

Nome del dataset	**%compressione**	**%accuratezza**	
		(1)	**(2)**
adultd	86,34	85,60	70,46
austrad	86,81	85,53	85,96
autod	73,33	72,86	71,43
breastd	93,93	95,38	95,38
cleved	82,41	71,15	71,15

diabetesd	78,66	73,28	73,28
germand	74,70	69,41	72,06
glassd	70,21	65,75	65,75
heartd	84,83	82,61	82,61
hepatid	86,27	83,02	81,13
horsed	85,95	81,75	81,75
ionod	89,61	81,67	81,67
irisd	90,91	100	100
letter-recognitiond	72,70	76,57	76,51
mushroom	99,65	100	99,78
pendigitsd	82,51	87,03	87
pimad	77,27	78,63	75,57
satd	83,85	84,78	85,01
segmentationd	89,44	93,89	93,89
sickd	97,51	98,21	98,32
sonard	83,21	83,10	83,10
soybean-large	79,21	84,76	81,90
soybean-s	87,10	93,75	37,50
splice	91,22	93,70	93,70
tic-tac	82,44	84,66	76,69
vehicled	75,99	67,36	67,36
voting	95,82	97,30	96,62
waveformd	76,36	74,53	74,82
wined	92,31	91,80	91,80
yeastd	57,41	59,60	59,60
zood	87,88	94,29	94,29

Tabella 4.6: Compressione ed accuratezza tramite Holdout

Nella colonna della Tabella 4.6 denominata “%accuratezza (1)” sono elencati i valori della percentuale di accuratezza ottenuta utilizzando il training set, costituito dal 66% delle istanze della collezione di dati iniziale, per la costruzione del modello di classificazione. Invece, per i risultati mostrati nella colonna “%accuratezza (2)”, il modello è stato generato dalla versione compressa delle stesse istanze di training utilizzate in precedenza. In entrambi i casi sono state utilizzate le stesse istanze di test costituite dal 33% delle istanze della collezione di dati non considerate come training set. Dal confronto dei valori ottenuti tra i due differenti test effettuati, è evidente che anche dopo la compressione non c’è una perdita elevata tra i valori di accuratezza, ma addirittura per alcuni dataset questi valori rimangono invariati. Da notare che ci sono dei casi particolari, poiché nel dataset denominato “adultd” il valore di accuratezza scende del 15% nel secondo test e nel dataset “soybean-s” la differenza tra i due valori raggiunge addirittura quasi il 60%. Considerando tutti e 31 i dataset analizzati, si ottiene un valore

medio del 84% prendendo in considerazione i valori della percentuale di accuratezza ottenuta utilizzando come dati di training il 66% delle istanze della collezione di dati iniziale. Invece, costruendo il modello di classificazione tramite la versione compressa delle istanze di training si ottiene una percentuale media del 81%. Quindi, in media si ha una perdita di accuratezza del 3% se si utilizza la versione compressa delle istanze di training.

4.7 Accuratezza di un modello per la compressione costruito dal 10%-15%-30% dei dati di training

In questo paragrafo verranno mostrati i risultati relativi ad ulteriori test svolti sui 31 dataset elencati nella Tabella 4.1. Inizialmente, una collezione di dati iniziale è stata divisa in due parti: 66% training set e le restanti istanze sono state considerate come test set. In seguito, il training set così generato è stato ulteriormente diviso in maniera differente per i tre diversi test effettuati. Infatti, l'idea è quella di considerare soltanto una parte del training set per generare un modello di classificazione utilizzato per la compressione. Questo test è utile poiché in presenza di grandi moli di dati risulta difficile o a volte impossibile la creazione di un modello di classificazione utile alla compressione a partire dall'intera collezione di dati iniziale. Inoltre, anche se si riuscisse a costruire il modello, talvolta questo potrebbe risultare troppo complesso. Per i tre differenti test effettuati sono stati considerati in un primo momento il 10% delle istanze del training set per generare il modello per la compressione, poi il 15% ed infine il 30%. Il modello di classificazione così generato viene utilizzato per effettuare la compressione di tutte le istanze del training set, costituite dal 66% delle istanze della collezione di dati iniziale. Infine, dalla versione compressa dei dati di training è stato generato un nuovo modello di classificazione che sarà utilizzato per classificare i dati di set costituiti dal 33% delle istanze della collezione di dati iniziale.

Nome del dataset	**%accuratezza**		
	modello 10%	**modello 15%**	**modello 30%**
adultd	70,65	69,48	69,54
austrad	77,45	77,45	84,26
autod	62,86	48,57	50
breastd	91,18	94,12	94,54
cleved	75,96	74,04	76,92
diabetesd	73,66	73,66	77,1
germand	68,53	68,53	68,82
glassd	58,9	60,27	61,64
heartd	67,39	72,83	79,35
hepatid	86,79	86,79	86,79
horsed	79,37	78,57	78,57

ionod	78,33	87,5	85,83
irisd	92,16	94,12	94,12
letter-recognitiond	66,38	67,4	70,54
mushroom	98,23	98,23	99,35
pendigitsd	77,96	80,2	82,13
pimad	75,19	80,92	78,63
satd	80,35	80,26	83,64
segmentationd	83,21	89,44	92,62
sickd	97,37	97,37	97,79
sonard	57,75	80,28	78,87
soybean-large	64,76	74,29	66,67
soybean-s	87,5	87,5	43,75
splice	78,61	84,17	90,65
tic-tac	66,56	75,15	75,15
vehicled	57,29	56,25	67,01
voting	97,3	97,3	97,3
waveformd	71,24	71,29	72,71
wined	68,85	77,05	83,61
yeastd	53,47	54,46	57,62
zood	82,86	77,14	88,57

Tabella 4.7: Confronto accuratezza modello di compressione 10%, 15% e 30%

Come si può notare dai valori mostrati nella Tabella 4.7, anche in questa tipologia di test è chiaro che le percentuali di accuratezza risultano alte, tranne per qualche caso particolare. In media tra i tre test effettuati è stato ottenuto un valore di accuratezza con percentuale pari al 77%. In aggiunta, dalla Tabella 4.7 si evince che aumentando il numero di istanze per generare il modello per la compressione si ottengono dei risultati leggermente superiori in termini di accuratezza dal modello di classificazione costruito dalla versione compressa delle istanze di training. Per valutare al meglio i risultati presenti nella Tabella 4.7 è necessario un confronto con uno dei risultati ottenuti nei test discussi nel Paragrafo 4.6. Infatti, è possibile effettuare un confronto tra i tre valori di accuratezza ottenuti dai tre esperimenti elencati in questo paragrafo e i risultati ottenuti dalla classificazione effettuata tramite Holdout, utilizzando le istanze di training non in versione compressa. I risultati del confronto sono mostrati nei Grafici 4.3, 4.4 e 4.5 dove si può notare che nella maggior parte dei casi questa differenza è minima. Di contro, per alcuni particolari dataset analizzati, come ad esempio per il dataset “autod” la differenza è grande in tutti e tre i grafici. Alcuni risultati anomali si ottengono dall’analisi dei dataset denominati “sonard” e “soybean-s”. Per il dataset “sonard”, la differenza è elevata nel Grafico 4.3, invece, nei Grafici 4.4 e 4.5 è minima. In maniera opposta si comporta il dataset denominato “soybean-s”, in cui nei Grafici 4.3 e 4.4 presenta una differenza minima, invece, nel Grafico 4.5 questa differenza è elevata.

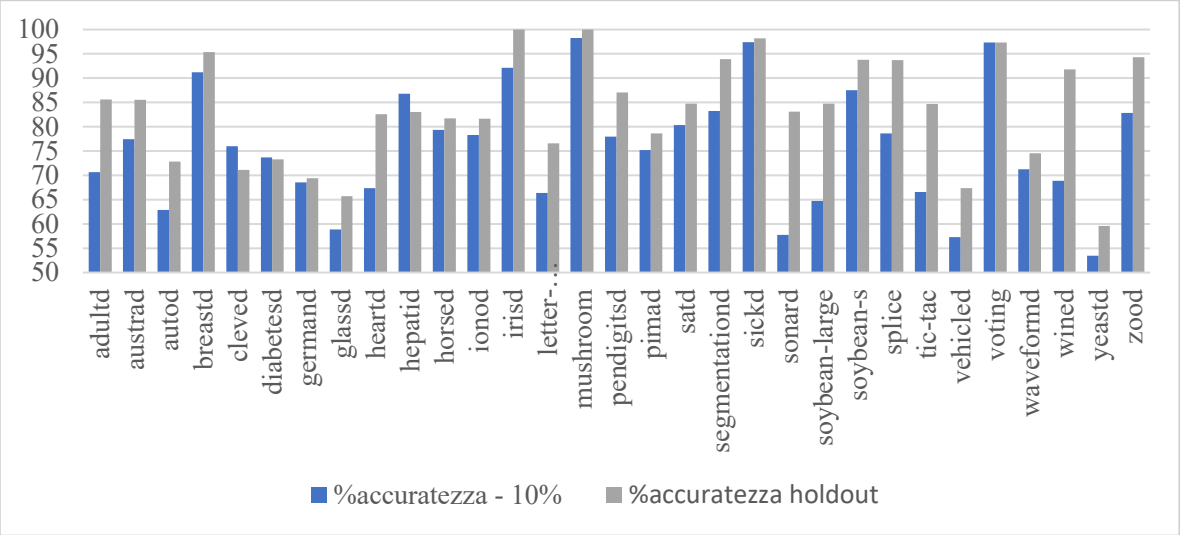


Grafico 4.3: Confronto accuratezza modello 10% e accuratezza Holdout

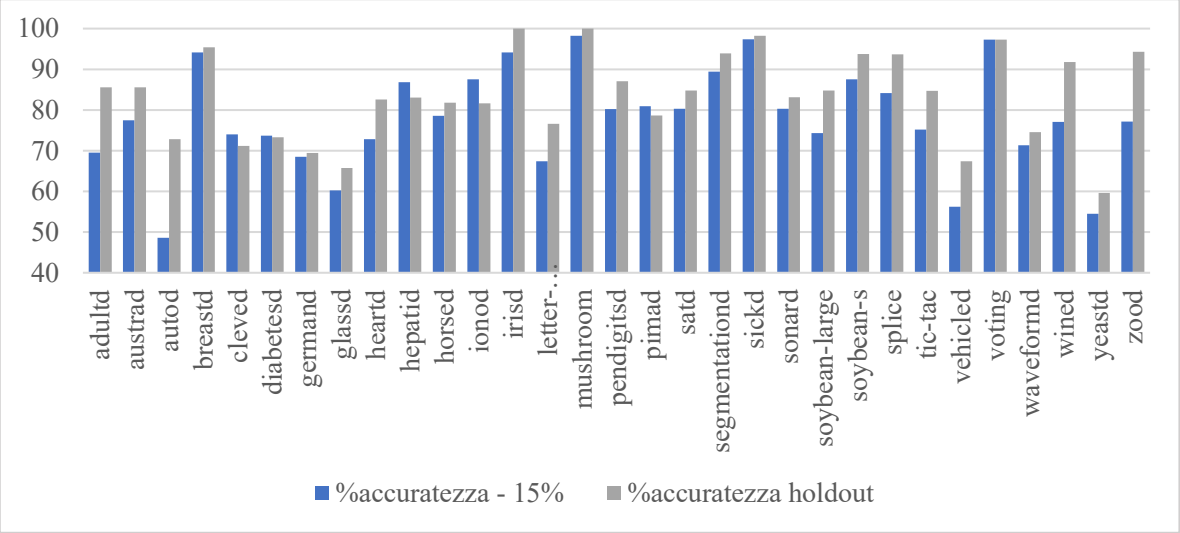


Grafico 4.4: Confronto accuratezza modello 15% e accuratezza Holdout

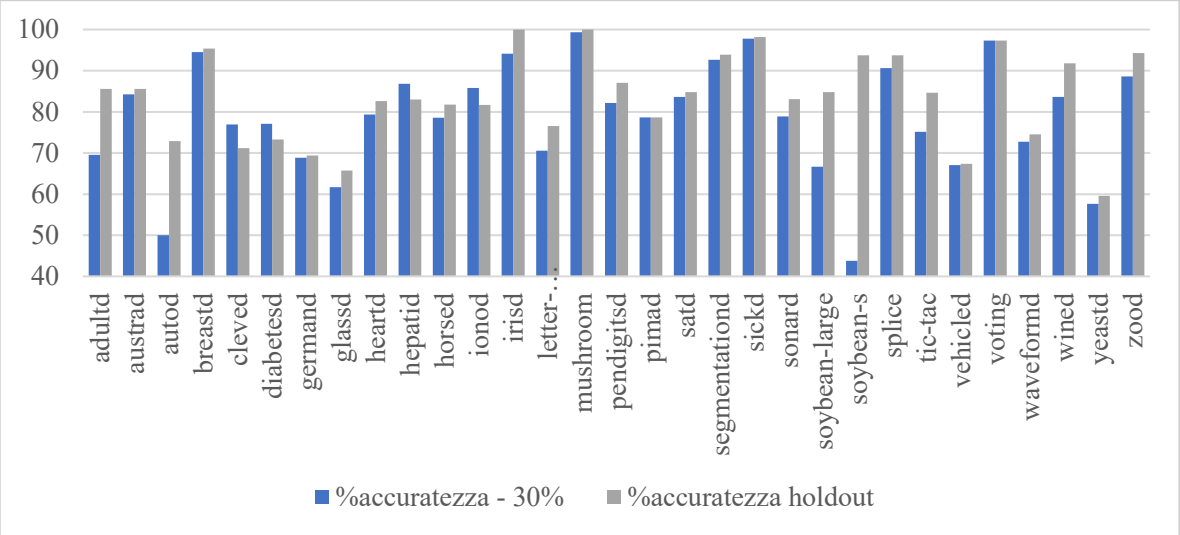


Grafico 4.5: Confronto accuratezza modello 30% e accuratezza Holdout

4.8 Test effettuati su un dataset di grandi dimensioni

In questo paragrafo verranno presentati i test svolti sul dataset di grandi dimensioni denominato "HIGGS" dell'UCI Machine Learning Repository. [6] Questo dataset è caratterizzato da 29 attributi numerici da un numero di istanze pari a 11 milioni. Data l'elevata cardinalità della collezione di dati, si è scelto di analizzare un sottoinsieme pari al 10% di queste istanze. Quindi, i risultati sono stati ottenuti sottoponendo ad analisi le prime 1,1 milioni di istanze del dataset "HIGGS".

4.8.1 Preprocessing del sottoinsieme del dataset

Innanzitutto, il dataset HIGGS dell'UCI Machine Learning Repository è stato convertito in un file in formato ARFF utilizzando il software Weka. Dato che questo dataset è costituito da attributi di tipo numerico, è stato necessario effettuare una fase di preprocessing per poterlo sottoporre ad analisi utilizzando la tecnica di classificazione basata sull'albero di decisione con algoritmo J48 di Weka. Quindi, è stata effettuata la discretizzazione di tutti gli attributi costituenti il dataset ad eccezione dell'attributo di classe. Per la discretizzazione si è scelto di dividere il dominio dei valori che un attributo può assumere in 20 intervalli. L'attributo di classe, invece, è stato convertito in nominale, visto che poteva assumere solo i valori numerici 0 e 1. Dopo la fase di preprocessing è stato possibile effettuare i test utilizzando l'algoritmo implementato in questa tesi.

4.8.2 Percentuale di compressione del dataset di grandi dimensioni

Un primo risultato utile da analizzare è il livello di compressione che si è ottenuto applicando l'algoritmo implementato in questa tesi al sottoinsieme delle istanze del dataset "HIGGS" costituito da 1,1 milioni di istanze (10% dell'intera collezione di dati). Per la costruzione del modello per la compressione è stato utilizzando il 66% delle istanze tra quelle del sottoinsieme del dataset iniziale, data l'elevata quantità di memoria necessaria dovuta all'eccessiva cardinalità della collezione di dati. Come dati di test per generare il dataset compresso, sono state utilizzate le stesse istanze del training set. Utilizzando questa configurazione è stato possibile ottenere una percentuale di compressione quasi del 70%, come mostrato in Tabella 4.8.

Nome dataset	n° istanze (66%)	n° percorsi	n° errori	%compressione
HIGGS (10%)	726000	14296	218705	67,91

Tabella 4.8 Compressione della collezione di dati di grandi dimensioni

4.8.3 Holdout del dataset di grandi dimensioni

In questo paragrafo sono mostrati i risultati di accuratezza ottenuti dai test svolti sul 10% delle istanze del dataset denominato "HIGGS". Inizialmente, le 1,1 milioni di istanze del sottoinsieme del dataset "HIGGS" considerato sono state divise in due parti: il 66% di queste sono state considerate come training set e le restanti come test set. Il primo test è stato effettuato costruendo il modello di classificazione tramite questo training set. Successivamente, un ulteriore test è stato svolto generando una versione compressa delle istanze di training iniziali, tramite la fase di compressione dell'algoritmo proposto in questa tesi. Il dataset compresso è stato generato applicando il 66% delle istanze del sottoinsieme del dataset "HIGGS" ad un modello per la compressione costruito tramite le stesse istanze. Successivamente, la versione compressa delle istanze di training è stata utilizzata per la costruzione del modello di classificazione. Per testare l'accuratezza della classificazione di entrambi i modelli di classificazione è stato utilizzato lo stesso test set costituito dal 33% delle istanze non considerate come dati di training.

Nome del dataset	n° istanze di training	n° istanze di training (compresse)	n° istanze di test	%accuratezza (1)	%accuratezza (2)
HIGGS (10%)	726000	233001	374000	66,86	70,11

Tabella 4.9 Confronto accuratezza Holdout per il dataset di grandi dimensioni

Nella colonna denominata "%accuratezza (1)" della Tabella 4.9 sono elencati i valori della percentuale di accuratezza ottenuta utilizzando il primo modello, costruito dal 66% delle istanze del sottoinsieme del dataset "HIGGS". Invece, per i risultati mostrati nella

colonna denominata "%accuratezza (2)", il modello è stato generato dalla versione compressa delle stesse istanze di training. In entrambi i casi sono state testate le stesse istanze di test costituite dal 33% delle istanze della collezione di dati non utilizzate come training set. La percentuale di accuratezza ottenuta dal modello di classificazione generato dalla versione compressa del 66% delle istanze del dataset "HIGGS" iniziale, ottenuta dall'applicazione della fase di compressione algoritmo implementato, è leggermente superiore rispetto a quello costruito con la sua versione originale. Infatti, si ha incremento in percentuale del valore di accuratezza del 4% poiché, grazie alla compressione applicata sulla collezione di dati, si riescono ad evitare istanze di rumore nella costruzione del modello di classificazione.

4.8.4 Accuratezza del modello utile alla compressione costruito dal 10%, 15% e 30% delle istanze di training

Per i test mostrati in questo paragrafo l'idea è quella di considerare soltanto una parte del training set per generare un modello di classificazione utile per la compressione, poiché in presenza di grandi moli di dati risulta difficile o a volte impossibile la creazione di questo modello partire dall'intera collezione di dati iniziale. Innanzitutto, il 10% del dataset "HIGGS" è stato diviso in due parti: 66% training set e le restanti istanze sono state considerate come test set. Successivamente, il training set così formato è stato suddiviso ulteriormente in tre modi differenti per effettuare tre test diversi. Per il primo test svolto è stato considerato soltanto il 10% delle istanze del training set per generare il modello per la compressione, poi sono state applicate a questo modello tutte le istanze del training set per la compressione. Dalle istanze compresse è stato generato il modello per la classificazione delle istanze di test.

Nome del dataset	n° istanze di training (66%)	n° errori	n° percorsi	%compressione	%accuratezza
HIGGS (10%)	726000	246362	3126	65,64	70,11

Tabella 4.10 Risultati modello generato dal 10% delle istanze di training

Nella Tabella 4.10 sono mostrati i risultati ottenuti dal primo test generando un modello di classificazione per la compressione dal 10% delle istanze di training. Nella colonna "%compressione" è mostrata la percentuale di compressione raggiunta grazie alla fase di compressione dell'algoritmo proposto applicata alle istanze di training. Nell'ultima colonna, invece, è elencato il valore di accuratezza ottenuto applicando le istanze di test, costituite dal 33% delle istanze iniziali, al modello di classificazione generato dalla versione compressa delle istanze di training.

Successivamente, è stato effettuato un secondo test considerando il 15% delle istanze del training set per la generazione del modello per la compressione. I risultati ottenuti da questo ulteriore test sono mostrati nella Tabella 4.11.

Nome del dataset	n° istanze di training (66%)	n° errori	n° percorsi	%compressione	%accuratezza
HIGGS (10%)	726000	241927	4445	66,10	70,11

Tabella 4.11 Risultati modello generato dal 15% delle istanze di training

In questo secondo test la percentuale di accuratezza rimane invariata, ma aumenta la percentuale compressione dato che vengono classificate un numero inferiore di istanze in modo errato dal modello.
Infine, un terzo test è stato svolto utilizzando il 30% delle istanze del training set per l'apprendimento del modello di classificazione utilizzato per la compressione di tutti i dati di training.

Nome del dataset	n° istanze di training (66%)	n° errori	n° percorsi	%compressione	%accuratezza
HIGGS (10%)	726000	232719	8279	66,80	70,11

Tabella 4.12 Risultati modello generato dal 30% delle istanze di training

Nella Tabella 4.12 sono mostrati i risultati ottenuti dal terzo test effettuato. La percentuale di accuratezza, confrontata con gli altri due test, rimane invariata, ma viene applicata una percentuale di compressione leggermente superiore rispetto agli altri due test effettuati.
A questo punto è utile il confronto tra i tre valori di accuratezza ottenuti da questi test con quello relativo alla tecnica di Holdout, mostrato nel Paragrafo 4.8.3, utilizzando il training set non compresso.

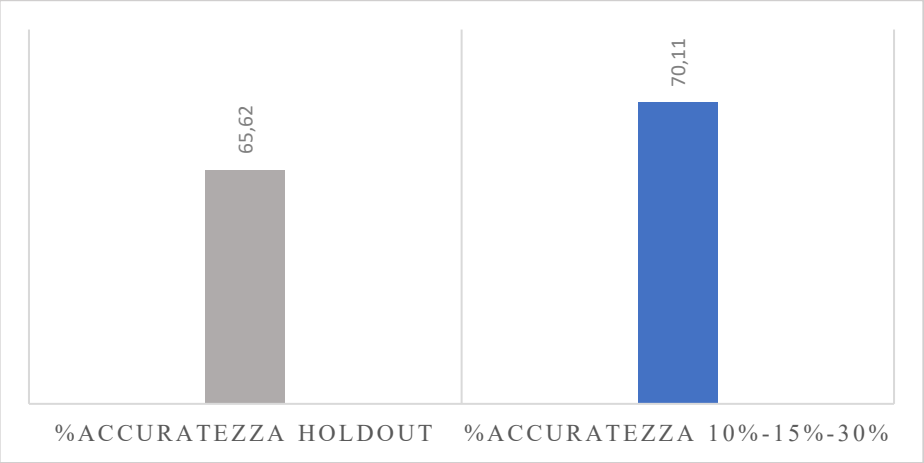


Grafico 4.6 Confronto accuratezza modello 10%-15%-30% con Holdout

Come si evince dal confronto tra le due percentuali di accuratezza mostrate nel Grafico 4.6, effettuando la compressione delle istanze di training con l'algoritmo implementato in questa tesi si ottengono dei risultati di accuratezza migliori. Infatti, si ha un aumento in percentuale del valore di accuratezza pari al 5%.
Un ulteriore test è stato effettuato per dimostrare che grazie alla compressione applicata sulla collezione di dati, si riescono ad evitare istanze di rumore nella costruzione del modello di classificazione. A questo proposito al modello utilizzato per la compressione sono state applicate le istanze di test costituite dal 33% delle istanze iniziali non utilizzate come dati di training. In quest'ultimo test svolto è stata ottenuta una percentuale di accuratezza del 66%. L'accuratezza data dal modello creato utilizzando il 66% dei dati senza compressione è simile a questo valore mentre l'accuratezza del modello generato dal 66% dei dati dopo la compressione è pari al 70%. Questo sta ad indicare che la compressione ha tolto del rumore, ma ha mantenuto l'informazione utile dei dati di training.

Capitolo 6
Conclusioni

In questo capitolo verranno tratte le conclusioni sul lavoro svolto durante lo svolgimento di questa tesi e verrà presentato un possibile sviluppo futuro.

6.1 Conclusioni

In questa tesi è stato implementato un algoritmo che ha lo scopo di comprimere una collezione di dati in modo che il modello di classificazione costruito a partire da questa versione compressa produca un livello di accuratezza simile al modello generato a partire dalla collezione di dati iniziale. L'utilità di questo algoritmo è evidente se si considerano le problematiche relative alla classificazione di collezioni di dati costituite da un'elevata dimensione, analizzate al Paragrafo 1.1. Infatti, per poter classificare dei dati di test occorre generare un modello di classificazione utilizzando dei dati di training etichettati. Per la costruzione di questo modello è necessario l'utilizzo di una certa quantità di memoria, che dipende dalla dimensione della collezione di dati iniziale, e talvolta o ne risulta impossibile la costruzione o può essere talmente complicato da produrre valori di accuratezza bassi. Per far fronte a queste problematiche di notevole importanza, si è scelto di implementare l'algoritmo proposto in questa tesi per comprimere la collezione di dati iniziale ed utilizzare questa versione compressa per generare il modello utile alla classificazione di istanze di test. Come mostrato nel Capitolo 4 dedicato alla parte sperimentale della tesi, l'algoritmo implementato è in grado di produrre un'elevata percentuale di compressione per tutte le collezioni di dati sottoposte ad analisi. Dato che l'obiettivo dell'algoritmo non è solo la compressione di grandi moli di dati, per un'analisi completa della sua qualità occorre valutare e confrontare i valori di accuratezza del modello di classificazione generato a partire dalla versione compressa di una collezione di dati con quello costruito utilizzando quella iniziale di grandi dimensioni. Sono stati effettuati varie tipologie di test per poter effettuare questo confronto e, in generale, è stato ottenuto un valore di accuratezza leggermente inferiore per il modello di classificazione costruito dalla versione compressa della collezione di dati. Di rilevante importanza assume il fatto che in un numero elevato di casi la differenza è pari a 0. Dato che una delle problematiche presenti è l'impossibilità della generazione del modello di classificazione in presenza di collezione di dati di grandi dimensioni, è stato necessario svolgere ulteriori test in cui sono stati generati due diversi modelli di classificazione: un primo modello è stato generato a partire dalle istanze di training. Invece, un secondo modello è stato costruito dalla versione compressa delle istanze di training iniziali, generata tramite l'applicazione della fase di compressione dell'algoritmo implementato in questa tesi in cui il modello di compressione è stato generato a partire da una porzione delle istanze di training. Tenuto conto dei risultati ottenuti per tutti i dataset analizzati, i

valori di accuratezza dal secondo modello di classificazione sono leggermente inferiori rispetto a quelli prodotti dal primo.
Dai risultati ottenuti da tutti i test effettuati, presentati nel Capitolo 4, è possibile trarre le conclusioni riguardo la qualità complessiva dell'algoritmo implementato in questa tesi. Infatti, dai livelli di compressione e di accuratezza ottenuti si può affermare la buona qualità dell'algoritmo proposto.

6.2 Sviluppi futuri

Per valutare un possibile sviluppo futuro occorre ricordare come l'algoritmo permette la generazione della versione compressa di una collezione di dati, ampiamente trattata nel Paragrafo 3.3.3.1. La fase di compressione dell'algoritmo implementato in questa tesi, infatti, trasforma i percorsi ottenuti in istanze nel formato iniziale utilizzando solo i valori degli attributi presenti nei percorsi. Ne consegue la creazione di istanze contenenti missing value, poiché i percorsi sono costituiti soltanto dagli attributi di split attraversati nel modello di classificazione utilizzato per la compressione. Dal test presentato al Paragrafo 4.4.1 è stato modificato il dataset compresso in modo da sostituire ai missing value il valore medio dell'attributo che li presentava. Questa scelta ha portato ad un'elevata perdita di accuratezza del modello di classificazione generato dal dataset compresso modificato rispetto al modello costruito utilizzando il dataset compresso con i missing value.
Come possibile sviluppo futuro, quindi, è proposto una scelta accurata dei valori per gli attributi delle istanze che presentano un valore mancante nel dataset compresso, poiché scegliendo valori opportuni si potrebbero ottenere percentuali di accuratezza superiori.

Bibliografia

[1] Prof. Elena Baralis, Sistemi per la gestione di basi di dati, [http://dbdmg.polito.it/wordpress/teaching/sistemi-per-la-gestione-di-basi-di-dati/].

[2] M. K. Jiawei Han, Data Mining: Concepts and Techniques, 2006

[3] Prof. Paolo Garza, Big Data: architectures and data analytics, 2017 [http://dbdmg.polito.it/wordpress/teaching/big-data-architectures-and-data-analytics-20162017/].

[4] Machine Learning Group at the University of Waikato, software Weka [http://www.cs.waikato.ac.nz/ml/weka/index.html]

[5] University of Waikato, WEKA Manual

[6] Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.