
POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

# An associative classification algorithm based on temporal association rules

**Relatore**

Prof. Paolo Garza

**Candidato**

Andrea Settimo

Ottobre, 2020

# Ringraziamenti

Prima di procedere nella trattazione del documento di tesi, desidero ringraziare il mio relatore Paolo Garza per la sua immensa disponibilità e chiarezza durante questo lavoro.

Un grazie particolare va alla Dott.ssa Pozzolo che mi ha dato utilissimi suggerimenti per condurre al meglio l'attività didattica universitaria.

Un doveroso ringraziamento va ai miei famigliari, ma soprattutto ai miei nonni: Rina, Armando e Iole; che mi hanno sostenuto durante il mio periodo di studi.

Ringrazio Andreina per avermi trasmesso la sua grande forza e il suo coraggio. Grazie per tutto il tempo che mi hai dedicato.

Infine, un profondo ringraziamento mia madre, mio padre e mia sorella che mi sono stati accanto per ogni piccola e grande difficoltà di questi anni.

# Indice

# Elenco delle figure

# Elenco delle tabelle

# Capitolo 1

# Introduzione

## 1.1 Obiettivo della tesi

Lo scopo di questa tesi è capire se un classificatore basato su regole di associazione possa essere utilizzato per predire situazioni basate sul concetto di tempo. Il motivo per cui si è scelto proprio un classificatore associativo è dato dal fatto che le regole di associazione permettono di estrarre condizioni di alta importanza presenti nel dataset di interesse, così facendo possiamo tenere in considerazione solo quelle situazioni che sono frequenti, necessarie per la predizione.

Per poter confermare la validità di questa tesi, il nostro studio deve essere basato su una tipologia di dataset che contenga il concetto di tempo. Il dataset di interesse è un file di log contente le informazioni relative alle stazioni di bike sharing della città di Barcellona, in cui sono contenuti gli stati delle stazioni, riguardanti il numero di biciclette disponibili e il timestamp di campionamento.

Come si può notare il dataset iniziale non è pronto per estrarre delle situazioni che permettano il training del nostro classificatore poiché lo stato di una relativa stazione è influenzato anche dalla sua situazione pregressa. Il dataset deve essere inizialmente modificato per raggiungere un nuovo formato, contenente le successive situazioni temporali delle stazioni. Una volta ottenuto questo dataset riadattato, bisognerà utilizzare un algoritmo di estrazione delle regole di associazione, estratte in modo opportuno durante la classificazione.

Il dataset di input è piuttosto grande per utilizzare un'implementazione del classificatore che estragga e compia predizioni con un classico paradigma dell'informatica. Per riuscire a classificare in modo performante e allo stesso tempo scalabile, la scelta opportuna è quella di utilizzare il paradigma suggerito dai Big Data, in questo documento verranno riportati metodologie di implementazione.

Il fine ultimo per raggiungere l'obiettivo è identificare la migliore configurazione o implementazione basata sul classificatore associativo, che permetta di predire lo stato delle stazioni.

Questo studio è utile, ad esempio per migliorare il servizio di distribuzione delle biciclette, in considerazione delle situazioni che si possono verificare in base agli eventi relativi alle stazioni di interesse. Il dato predittivo può fornire informazioni tali da identificare le stazioni che in futuro saranno sprovviste di biciclette e rifornirle, renderle disponibili all'utenza con un notevole ritorno economico.

## 1.2 Struttura del documento

Il documento di tesi è strutturato in cinque capitoli organizzati nel seguente modo:

- **Capitolo 2:** presentazione dei concetti generali che permetto al lettore di capire cosa c'è dietro a livello teorico allo sviluppo delle soluzioni proposte;
- **Capitolo 3:** descrizione e svolgimento dello sviluppo, soluzione con relativi esempi e immagini che permetto di semplificare la comprensione;
- **Capitolo 4:** descrizione di vari esperimenti svolti, utilizzando le soluzioni proposte;
- **Capitolo 5:** esposizione delle conclusioni con relativi accenni a possibili sviluppi futuri del documento.

# Capitolo 2

# Concetti generali

In questo capitolo verranno presentati i concetti teorici che sono alla base di questa tesi e l'ambiente di sviluppo utilizzato. I concetti esposti riguardano l'utilizzo del Data Mining e il supporto dei Big Data per poter compiere il task definito nell'introduzione.

## 2.1 Data Science

La Data Science [5] è una disciplina che permette l'estrazione di un nuovo significato da una grande quantità di dati. Essa è composta da molte aree:

- Parte analitica definita da:
  - Data Mining;
  - Machine Learning;
  - Pattern Recognition;
- Statistica;
- Visualization;
- Data Processing;
- Neuro Computing.

Il processo di Data Science è composto dalle seguenti fasi (figura 2.1):

- Generazione dei dati: esistono differenti fonti di generazione, dove possiamo identificarne diverse possibili modalità:
  - Modalità passiva: generazione di dati di tipo strutturato, definita ad esempio transazione bancarie, record derivati da acquisti, ...;
  - Modalità attiva: generazione di dati di tipo semi strutturato o non strutturato. Ad esempio, dati generati da utenti su una piattaforma di social network, ...;
  - Modalità automatica: generata da sistemi che si interfacciano con sensori;

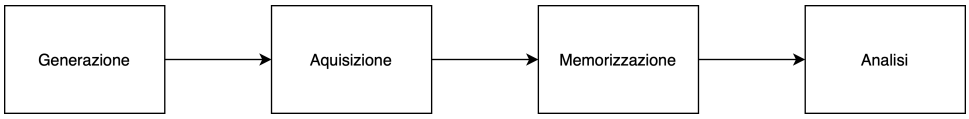


*Figura 2.1: Processo della Data Science*

- Acquisizione dei dati: una volta generati bisogna acquisirli e trasmetterli verso ad un data center, dove sarà possibile effettuare il pre-processing, attraverso la manipolazione che permetterà di effettuare il cleaning ed eliminare l'informazione ridondante o inutile;
- Memorizzazione dei dati: inserimento all'interno di un database per tenerli in modo persistente con una modalità per la loro gestione;
- Analisi: insieme di diversi processi che hanno il fine ultimo di estrarre la conoscenza dai dati, inoltre è necessario identificare qual è il tipo di analisi più adatta ai nostri dati.

## 2.2 La sfida dei Big Data

Con l'avvento degli apparati di acquisizione dati (es. sensori termici, dati finanziari, ...) che restituiscono una grande mole di dati, è necessario avere delle procedure che permettano di analizzarli in modo efficiente. La sfida colta dal Big Data consentirebbe di operare su una grande quantità di dati in modo efficiente. Le problematiche relative alla grande quantità di dati sono:

- Velocità: i dati vengono generati in modo molto veloce (es. campionamento dei sensori che avviene ad intervalli dell'ordine di millisecondi);
- Processamento dei dati in real-time (streaming di dati): dati che arrivano da più sorgenti, utilizzati contemporaneamente per effettuare predizioni in tempo reale per comprendere una determinata situazione;
- Varietà: i dati prodotti di diverso formato, tipo e struttura;
- Valore: può essere di vario tipo, riuscire ad analizzare i dati e trasformali in un vantaggio, ad esempio può essere migliorare un determinato servizio di una compagnia o generare utile finanziario.

La gestione delle problematiche sono risolte grazie all'utilizzo di nuove architetture, paradigmi di programmazione e tecniche che permettono di riuscire a gestire in modo performante e semplice una grande quantità di dati. Due possibili framework di Big Data sono: MapReduce e Spark.

## 2.3 Data Mining

Il Data Mining [6] è una tecnica di estrazione di informazioni implicite, sconosciute e potenzialmente utili provenienti da dati già disponibili. L'estrazione viene fatta in modo automatico mediante l'utilizzo di un algoritmo. L'informazione risultante viene rappresentata attraverso un pattern di dati.

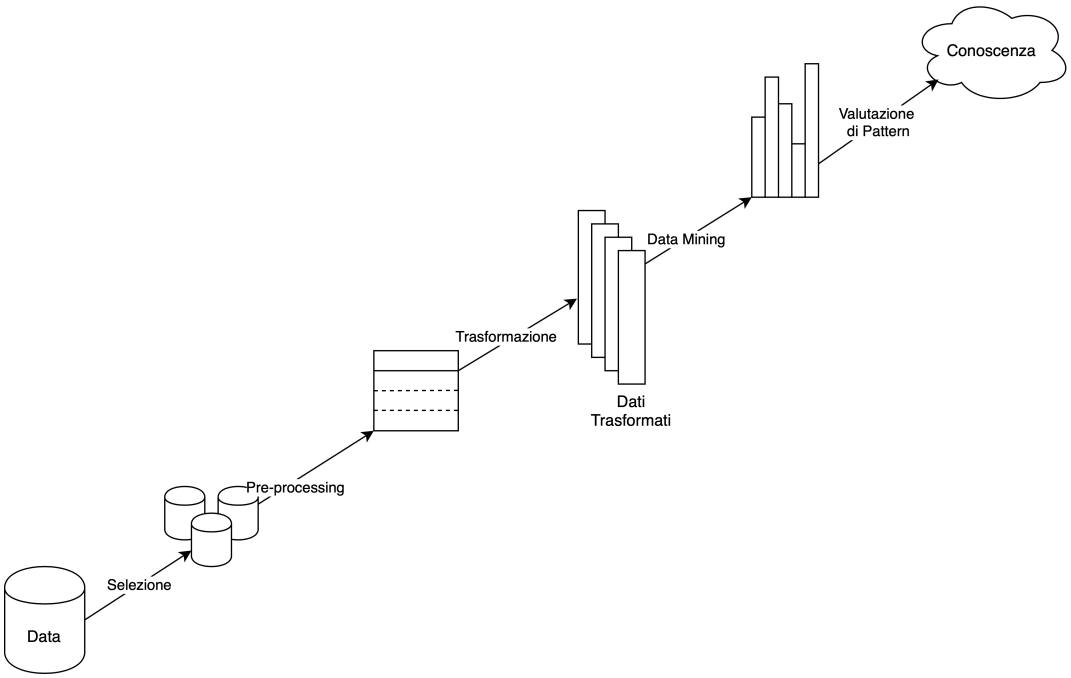


*Figura 2.2: Knowledge Discovery from Data*

## 2.4 Knowledge Discovery from Data

Un task di Data Science è descritto dal processo di KDD (Knowledge Discovery from Data)(figura 2.2) [5], costituito da diverse fasi:

- Selezione dei dati su cui operare;
- Pre-processing: sono una serie di tecniche che permettono di manipolare i dati in ingresso in base alla loro struttura per aumentarne la qualità. Nel caso in cui si riesca ad ottenere una buona qualità di dati, in uscita si avranno dei buoni pattern. Un esempio di approccio è feature reduction: permette di diminuire la dimensionalità del nostro dataset;
- Trasformazione dei dati: al fine di semplificare l'analisi;
- Data mining: utilizzare un algoritmo di data mining per poter estrarre l'informazione celata nei dati;
- Interpretazione: ci dovrà essere una persona qualificata nel dominio di analisi per poter confermare la "bontà" dei risultati ottenuti.

Le macro-tipologie di problemi affrontati nell'ambito del Data Mining [6] sono le regole di associazione, la classificazione e il clustering. Le regole di associazione, come descritto nel paragrafo successivo, descrivono situazioni di interesse all'interno del nostro dataset, invece la classificazione permette di classificare i dati sulla base di una conoscenza pregressa. Il clustering è una tecnica che permette di capire come sono relazionati/raggruppati tra di loro i dati presenti nel dataset, in questo caso non c'è bisogno di una conoscenza pregressa per attribuire un determinato dato ad un cluster.

| N. Transazione | Item |
|---|---|
| 1 | {a,d,e} |
| 2 | {a,b,c,e} |
| 3 | {a,b,d,e} |
| 4 | {a,c,d,e} |
| 5 | {b,c,e} |
| 6 | {b,d,e} |
| 7 | {c,d} |

*Tabella 2.1: Dataset di esempio.*

## 2.5 Regole di associazione

Le regole di associazione [7] permettono di estrarre situazioni di correlazione tra oggetti all'interno di un dataset. Il dataset è una collezione di transazioni. Le transazioni sono un insieme di item.

Nella tabella 2.1 rappresenta un esempio di dataset, composta dal numero di transizione e gli item associati ad esso.

Possiamo definire una regola nel seguente modo:

$$A, B \rightarrow C$$

Dove A e B sono il corpo della regola (anche chiamata antecedente) e C è la testa della regola (anche chiamata conseguente). Questa rappresentazione non è un'implicazione logica. Le regole di associazione costituiscono una tecnica di esplorazione che è possibile applicare su ogni tipo di dato. Di conseguenza possiamo definire che questa tecnica non vuole effettuare perdizioni sui dati, ma serve per capire come è composto il nostro dataset.

### 2.5.1 Definizioni

In generale, la testa e il corpo della regola sono degli itemset. Possiamo definire un elenco di termini utili per migliorarne la comprensione:

- Itemset: è un insieme di uno o più item;
- K-itemset: è la cardinalità dell'itemset;
- Supporto: è il rapporto del numero di transizioni che contengono un itemset diviso il numero totale di transizioni:

$$sup(A) = \frac{frequenza(A)}{\#\ totale\ di\ tranzioni}$$

- Itemset frequenti: sono itemset che hanno supporto maggiore di un valore di soglia chiamato minimo supporto (Minsup);
- Confidenza: è la probabilità condizionale di trovare B dato A:

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

| Items | freq |
|---|---|
| [e] | 6 |
| [d] | 5 |
| [d, e] | 4 |
| [a] | 4 |
| [a, e] | 4 |
| [b] | 4 |
| [b, e] | 4 |
| [c] | 4 |

*Tabella 2.2: Itemset frequenti estratti con soglia di minimo supporto pari a 0.5.*

| regole | confidenza | lift |
|---|---|---|
| b→e | 1.0 | 1.1667 |
| a→e | 1.0 | 1.1667 |
| d→e | 0.8 | 0.9333 |
| e→d | 0.6667 | 0.9333 |
| e→a | 0.6667 | 1.1667 |
| e→b | 0.6667 | 1.1667 |

*Tabella 2.3: Regole di associazione estratte con valore di confidenza minima pari a 0.5.*

- Lift: utilizzando la confidenza come misura di estrazione si possono verificare dei problemi nel caso in cui la testa della regola sia molto frequente. Per risolvere questo problema si può utilizzare un altra misura chiamata Lift, definita nel seguente modo:

$$lift = \frac{P(A|B)}{P(A)P(B)} = \frac{conf(A \rightarrow B)}{sup(B)}$$

### 2.5.2 Estrazione delle regole

Per estrarre delle regole bisogna imporre dei vincoli sulle misure utilizzate nella generazione:

- Supporto minimo (MinSup): gli itemset generati devono avere il valore di supporto:

$$supporto \geq MinSup$$

- Confidenza minimo (MinConf): Definire come il minimo valore di confidenza che gli itemset devono avere.

$$conf \geq MinConf$$

Nelle tabelle 2.2 e 2.3 è possibile vedere gli itemset frequenti e le regole estratte imponendo come vincolo di minimo supporto pari 0.5 e il minimo di confidenza pari a 0.5.

### 2.5.3 Tecniche di estrazione

#### Brute Force

Un possibile algoritmo di generazione di itemset frequenti può essere quello del Brute Force. Questa è la tecnica più "semplice" ma con il costo di computazione più alto, consiste nell'enumerare tutte le possibili permutazioni degli item, così facendo da creare tutti i possibili itemset. Il costo di questa tecnica è:

$$O(|T|2^d w)$$

Dove:

- $|T|$: rappresenta la cardinalità delle transizioni;
- $D$: il numero di item;
- $w$: lunghezza della transizione.

#### Tecniche alternative

Per evitare di utilizzare la tecnica del Brute Force si possono utilizzare delle alternative, si basano sul generare prima gli itemset frequenti, imponendo un vincolo sul supporto e/o sulla confidenza, utilizzando tecniche come: Apriori e FP-Growth. Come secondo passo, si possono generare tutte le possibili regole di associazione dagli itemset frequenti. Queste tipologie di tecniche permetto di ridurre lo spazio di ricerca, con la possibilità di:

- Ridurre il numero di candidati;
- Ridurre il numero di transizioni;
- Rendere efficiente il confronto.

#### Apriori

Questa tecnica si basa sul seguente principio:

*Se un itemset è frequente, allora anche i suoi sottoinsiemi sono frequenti.*

Nel rovesciare questa condizione si può ottenere un'altra tipologia di condizione:

*Se un certo insieme non è frequente anche i suoi sopra-insiemi non sono frequenti.*

Questo implica che il supporto di un itemset non può essere più elevato di un suo sottoinsieme:

$$A \subseteq B \rightarrow sup(A) \geq sup(B)$$

Questo algoritmo si basa su due passi fondamentali, effettuati ad ogni iterazione nel valutare l'estrazione di itemset frequenti di lunghezza pari a k:

- Generazione dei candidati: questa fase è composta da altri due sottopassi:
  - Self-join dei candidati che hanno in comune il prefisso di lunghezza pari k-1, presenti nell'insieme $L_{k-1}$;

  - Applicazione del principio su cui si basa questo algoritmo, eliminazione dei candidati se almeno uno dei suoi sottoinsiemi non è contenuto in $L_{k-1}$. Questa fase viene chiamata pruning dei candidati;

- Generazione degli itemset frequenti andando a prendere solo in considerazione i candidati che soddisfano il vincolo sul minimo supporto. Questo implica scansionare tutto il dataset per calcolare il supporto dei candidati.

### FP-Growth

L'approccio è quello di leggere il dataset in memoria in un colpo solo, usando una rappresentazione compressa e leggendolo in modo ricorsivo utilizzando una struttura dati chiamata FP-tree, che è una completa rappresentazione degli itemset frequenti (come quella dell'approccio Apriori), che non contiene gli item aventi un valore di supporto inferiore rispetto alla soglia.

L'algoritmo consiste:

1. Calcolo del supporto degli item, applicando la condizione sul minimo supporto;

2. Costruzione della header table: tabella contenente la corrispondenza degli item con il loro relativo supporto associato, ordinata per supporto decrescente;

3. Creazione del FP-tree, per ogni transazione:

   (a) Ordinare gli item in base all' header table;

   (b) Inserire gli item usando lo stesso percorso con i prefissi comuni, nel caso se ne crea uno nuovo.

### Confronto tra Apriori e FP-Growth

L'algoritmo Apriori ha molte problematiche relative alle prestazioni:

- Leggere più volte il dataset per contare il supporto degli itemset, questo può essere un problema quando il numero di candidati è elevato, in una determinata transazione vi possono essere molti item;

- La generazione dei candidati può essere molto onerosa, perché nella generazione bisogna effettuare il prodotto scalare di L1 con sé stesso;

- L'estrazione di intemset lunghi e frequenti richiede la generazione di tutti i sottoinsiemi frequenti.

Nel caso dell'algoritmo di FP-Growth, come detto in precedenza, l'utilizzo della struttura dati FP-tree permette di rappresentare i dati in una versione compressa del dataset. La creazione di questa struttura dati non genera nessun candidato e comporta un notevole miglioramento delle prestazioni rispetto all'algoritmo Apriori.

### 2.5.4 Effetti sulla scelta della soglia di supporto

Un altro problema è quello di determinare un'opportuna soglia di supporto, in generale si può riscontrare:

- Supporto troppo alto: si ottengono itemset troppo frequenti, che potrebbero essere già noti, e si possono perdere item che hanno una forte correlazione tra di loro ma con una bassa frequenza (che è il nostro caso di interesse);
- Supporto troppo basso: porta ad avere troppe transizioni da valutare e difficoltà nel parallelizzare.

## 2.6 Estrazione di sequenze frequenti

L'estrazione di sequenze frequenti è un problema noto nel Data Mining, vi sono molti algoritmi che permettono l'estrazione di sequenze. Molti di questi algoritmi si basano sul procedimento dell'algoritmo Apriori, come ad esempio GSP. Esistono altri algoritmi come FreeScan e SPADE, che cercano di migliorare le prestazioni di estrazione dell'algoritmo GSP, poiché eredita le stesse problematiche dell'algoritmo Apriori. Una sequenza è costituita da una lista di eventi denotati con $\alpha$:

$$\alpha_0 \rightarrow \alpha_1 \rightarrow \ldots \rightarrow \alpha_n$$

### 2.6.1 FreeScan

FreeScan (Frequent Pattern-Projection Sequential Pattern Mining) [1] è basato sui seguenti passi:

1. Viene scansionato il dataset contenente le sequenze di interesse, e viene costruita la f_list (cioè la lista degli item frequenti ordianta in modo decrescente);
2. Viene eseguito alternative_level projection:
   (a) Costruzione della matrice F;
   (b) Generazione delle sequenze di lunghezza 2 frequenti, con l'insieme di proiezione del dataset e le annotazioni relative alle ripetizioni degli item;
   (c) Viene scansionato il dataset per la costruzione degli item ripetuti e dalla proiezione del dataset;
   (d) Costruzione della matrice di proiezione sul dataset;
   (e) Continua al punto b, fino a quando non vi sono più dei candidati.

### 2.6.2 SPADE

SPADE (Sequential PAttern Discovery using Equivalence classes) [16] permette di generare le sequenze frequenti effettuando i seguenti macro passi:

1. Generare le sequenze frequenti di lunghezza pari a 1;
2. Generare le sequenze frequenti di lunghezza pari a 2;

3. Successivamente si genera per ogni lettera la classe di equivalenza;

4. Per ogni classe di equivalenza si generano tutte le sequenze frequenti generate partendo da un join di tutti gli "atomi" presenti (cioè sequenze di una determinata lunghezza che sono frequenti) e applicando per ognuno un vincolo di pruning: scartando tutte le lettere già processate; in successiva, in modo ricorsivo si può decidere di rieseguire (4), applicando due strategie di ricerca: BFS (Breadth-First Search) o DFS (Depth-First Search).

Questi due algoritmi (FreeScan e SPADE) in questa tesi non sono direttamente applicabili per poter estrarre le sequenze temporali, per questo motivo si è deciso di estrarre gli itemset frequenti e le regole di associazione utilizzando l'algoritmo FP-Growth.

## 2.7 Classificazione

Dato un dataset composto da dati (rappresentati dalla lettera X) e la loro etichetta di classe (rappresentata dalla lettera Y), possiamo redigere un modello che permetta di predire la classe di dati non conosciuti.

La Y deve essere di tipo categorico, che ci permetta di caratterizzare i nostri dati con una determinata classe per effettuare la predizione su di essi.

### 2.7.1 Classificatore associativo

Il classificatore associativo [4] permette di generare un modello basato sulle regole di associazione, descritto dalla seguente formulazione:

$$Corpo\ della\ regola\ (condizione)\ \rightarrow\ y$$

Questo tipo di classificatore si basa sui seguenti passi:

- Generazione delle regole di associazione: utilizzando un algoritmo per l'estrazione di item set frequenti in base ad una relativa soglia di supporto, confidenza o correlazione;

- Verifica che le regole vanno a ricoprire il nostro dataset.

Come si può evincere dalle regole di associazione, questo modello può essere molto accurato perché vengono considerate le correlazioni tra gli attributi del dataset e l'efficienza nella classificazione.

Tuttavia, si possono generare dei problemi relativi alla generazione delle regole di associazione che può risultare lenta e dipende dai valori di soglia.

### 2.7.2 Divisione del dataset: training e test set

Per effettuare un task di classificazione c'è bisogno di suddividere il nostro dataset in due porzioni. Queste due porzioni sono definite nel seguente modo:

- Training set: è la parte del dataset su cui andremo a creare il nostro modello di classificazione;

- Test set: è la parte di dataset che il nostro modello che non dovrà valutare fino al temine del training, permette di capire la qualità del nostro modello.

Il problema principale è che non si conosce a priori la proporzione da attribuire ai due sottoinsiemi, per questo si può ricorrere all'utilizzo della learning curve. Il learning curve permette di valutare diversi valori di grandezza del training set e di test set, valutando le relative performance del tipo di classificatore preso in esame.

Al termine, si decide qual è il migliore split in accordo alle migliori performance che si possono avere sul test set.

La figura 2.3 mostra il processo di divisione del dataset e i passi concettuali della classificazione.

### Scelta degli hyperparameters

Molti classificatori sono basati su parametri che lo sviluppatore può impostare, prendono il nome di "hyperparameters". Gli hyperparameters sono valori intrinseci dell'algoritmo su cui il nostro classificatore si basa. Il problema principale è che non sappiamo quali sono i migliori hyperparamiters da scegliere e per questo motivo necessita una analisi accurata su quali valori selezionare.

La decisione sulla selezione degli hyperparameters può essere fatta utilizzando la validazione. La validazione si basa sul concetto di valutare il relativo modello utilizzando una porzione di training set, chiamato validation set, su cui valutare le performance. Nel nostro caso però, non abbiamo abbastanza dati per effettuare la validazione con un validation set dedicato, quindi andremo ad utilizzare direttamente il test set per la scelta di essi.

L'analisi per la scelta di quale set di hyperprameters selezionare può essere ricondotta a due possibili tecniche:

- Grid search: selezionare dei valori prefissati su cui effettuare la validazione. Tecnica molto rigida che comporta la non valutazione di valori hyperparameters che potrebbero lavorare "bene" sul nostro modello;

- Random search: questa tecnica va a rimediare al problema relativo alla tecnica precedente. In questo caso, si introduce il concetto di randomicità. Dove ad ogni iterazione di valutazione si va ad effettuare un campionamento all'interno di un range di valori pari agli hyperparmeters. Possiamo attribuire a questo range di valori una distribuzione uniforme, cosicché tutti i valori possibili abbiano la stessa probabilità di essere campionati. Questa metodologia permette di aumentare lo spazio di ricerca dei parametri da selezionare.

Una volta finita questa fase si selezioneranno i migliori valori di hyperparmeters con il relativo modello. Invece, nella classica procedura di validazione si dovrà effettuare il training su tutto il training set con i valori di hyperparemeters selezionati.

## 2.8 Metriche

Per poter capire la qualità dei nostri risultati possiamo utilizzare delle misure che vanno a confrontare le classi predette con le classi reali del test.

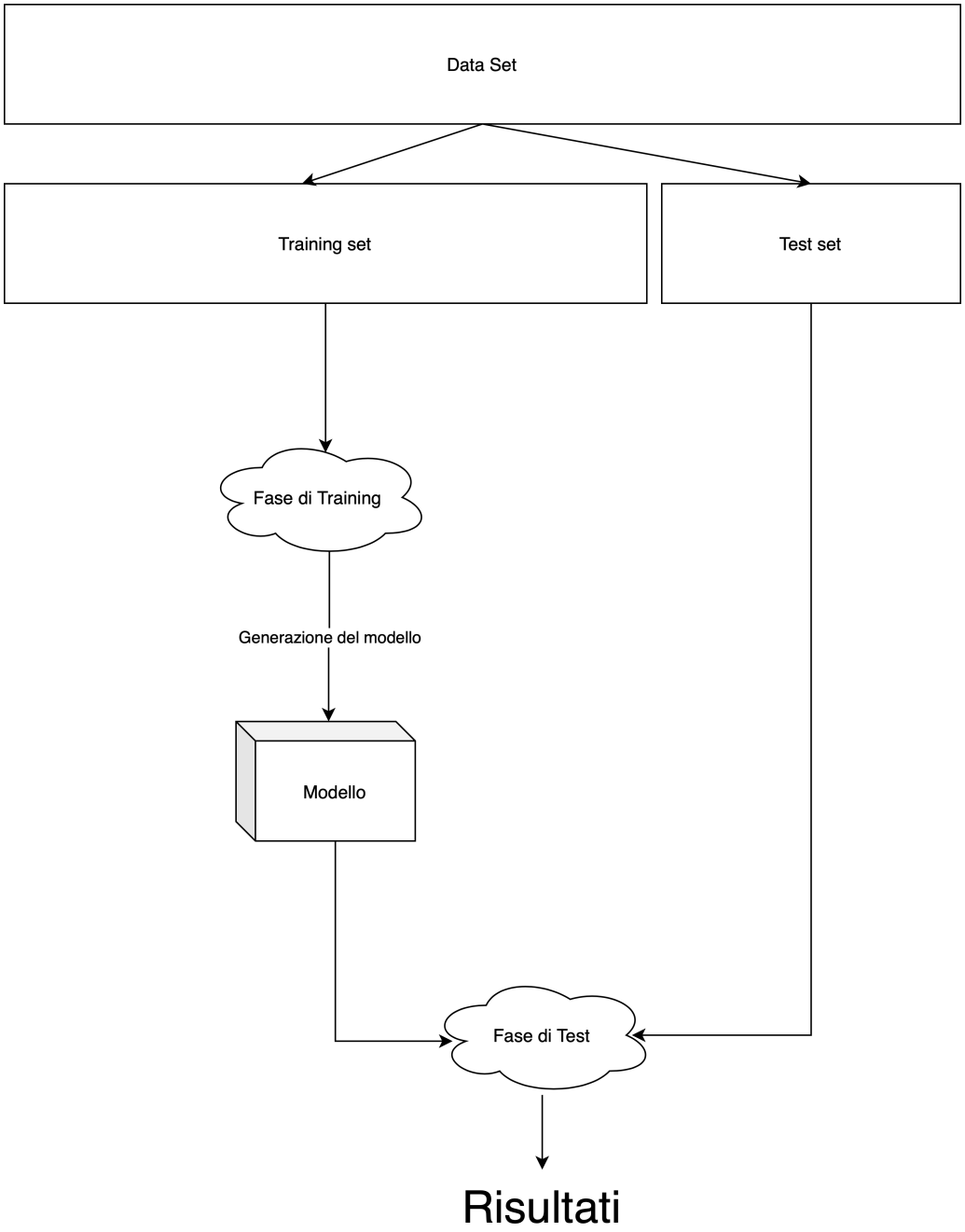


*Figura 2.3: Divisione del Dataset e classificazione*

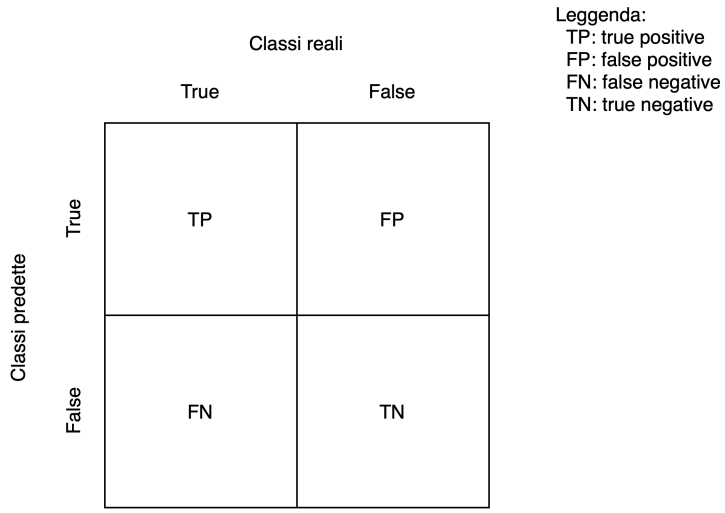


*Figura 2.4: Matrice di Confusione*

### 2.8.1 Matrice di confusione

La matrice di confusione (figura 2.4) [4] è strutturata nel seguente modo:

- Sulle colonne vi sono le classi reali del nostro test set;
- Sulle righe vi sono le classi predette del nostro test set.

Ogni cella rappresenta la frequenza dei dati di come sono stati classificati in accordo con la classe di predizione e con la sua reale classe.

Sulla diagonale, come si può notare, si hanno tutte le predizioni corrette, fuori dalla diagonale ci sono gli oggetti etichettati in modo sbagliato.

### 2.8.2 Accuratezza

Una prima conseguenza della matrice di confusione è quella di ottenere la misura di accuratezza [4] che va a definire quanto il nostro modello è accurato sul test set. L'accuratezza è descritta dalla seguente formula:

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

Tuttavia, l'accuratezza non va bene quando si hanno classi sbilanciate, poichè privilegia la classe dominante e non permette di distinguere i casi di errore più importanti e da quelli meno importanti.

### 2.8.3 Richiamo

Il richiamo [4] va a definire la percentuale di dati di test che sono all'interno di una determinata classe, descritta dalla seguente formula:

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

### 2.8.4 Precisione

La precisione [4] permette di andare a quantificare quanti elementi di una classe sono rilevanti, descritta dalla seguente formula:

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

### 2.8.5 F1 Score

Il richiamo e la precisione, non possono essere massimizzate contemporaneamente, allora si può effettuare una media armonica di essi, per ottenere una nuova misura chiamata $F_1$ score [4], definita nel seguente modo:

$$F_1 score = 2 \cdot \frac{Precisione \cdot Richiamo}{Precisione \ + \ Richiamo}$$

## 2.9 Ambiente di sviluppo

### 2.9.1 Spark

Apache Spark [3] è un framework open source per il Big Data che permette di effettuare in modo semplice calcoli distributi su calcolatori diversi. È caratterizzato dalle seguenti proprietà:

- Mantiene i dati in memoria principale per ottenere delle prestazioni migliori rispetto al framework MapReduce, quest'ultimo per compiere i task deve accedere alla memoria secondaria. Per questo motivo Spark cerca di mantenere tutti i dati in memoria principale, per abbattere il costo di accesso per memorizzare/caricare i dati da memoria secondaria;

- Cerca di essere il più generico possibile riguardante il carico di lavoro assegnato ai workloaders;

- Mette a disposizione allo sviluppatore un insieme di API (Application Programming Interface) su diversi linguaggi di programmazione (Java, Python, Scala e R) a differenza di MapReduce che permette lo sviluppo soltanto su Java;

- L'esecuzione avviene in modalità indipendente ed ha una buona integrazione con Hadoop YARN Cluster Manager;

- L'utilizzo del livello di astrazione dato dagli RDD permette di essere fault tollerance.

#### Componenti di Spark

Spark [9] è costituito da una serie di componenti, che permettono allo sviluppatore di compiere differenti tipi di task:

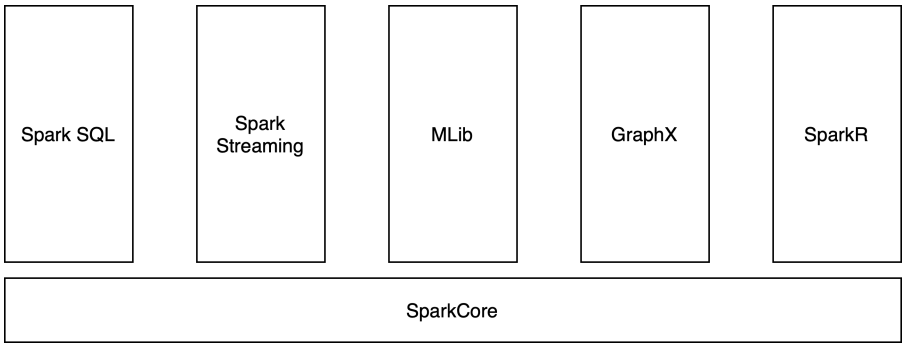


*Figura 2.5: Spark framework*

- Spark SQL: permette di effettuare delle operazioni mediante dati strutturati e utilizzare metodi per effettuare operazioni simili a query, presenti nel mondo SQL. Per poter accedere a queste funzionalità bisogna creare un DataFrame partendo da un RDD. Inoltre, vi è un ottimizzatore che va ad ottimizzare le operazioni che vengono eseguite;
- Spark Streaming: premette di operare su dati che arrivano in un flusso continuo, generati anche in real time. Questo flusso di informazione può arrivare da una serie di sorgenti diverse, come ad esempio Kafka, Flume, Kinesis e socket TCP;
- MLib: è una libreria dedicata al Machine Learning, dove vi sono tutte le implementazioni riguardanti i principali algoritmi. All'interno di questa libreria vi è anche l'implementazione dell'algoritmo di FP-Growth [8];
- GraphX: permette di compiere la computazione per la risoluzione di problemi basati sui grafi, poiché molti algoritmi si basano sul trovare il percorso migliore all'interno di un grafo;
- SparkR: questa libreria è disponibile solo per R, basta su SparkR DataFrame che permette in modo semplice la manipolazione dei dati, la computazione e la visualizzazione grafica.

Questi componenti (figura 2.5) si basano tutti su un altro layer di nome SparkCore, che ha la capacità di eseguire in parallelo e in modo distribuito la computazione dei dati sul cluster.

### Struttura

Per poter accedere alle funzionalità messe a disposizione da Spark, bisogna utilizzare lo SparkContext che consente all'applicazione di accedere al Cluster con l'aiuto del Resource Manager, come si evince della figura 2.6.

Il resource managar interagisce con i worker node, l'esecuzione viene demandata agli executors. Ogni executor esegue un task su una piccola porzione di dati definita all'interno del RDD, interagisce con il filesystem locale o distribuito.

### Hadoop File System

Hadoop File System (HDFS) è un file system distribuito, progettato per essere fault tollerance e che utilizza hardware di basso costo.

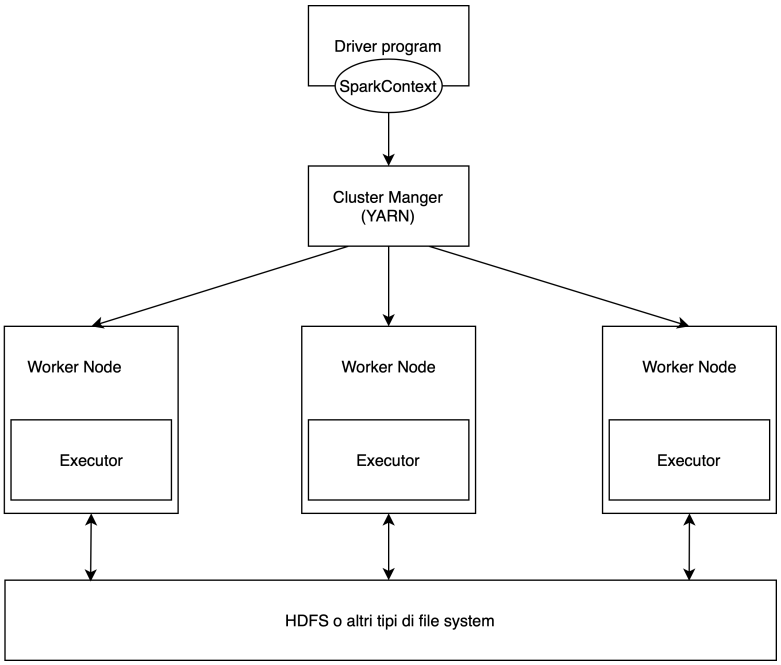


*Figura 2.6: Architettura di Spark*

Questa tipologia di file system permette di memorizzare grandi quantità dati. La memorizzazione avviene su più macchine in modo ridondante per prevenire problemi di perdita di dati in caso di guasto. Questo tipo di approccio, di conseguenza, permette a più calcolatori di compiere l'esecuzione in parallelo per migliorare le performance.

### RDD

L'RDD [10] è un particolare oggetto, la cui rappresentazione è un set di dati partizionati sugli executors. Le operazioni eseguibili sono: trasformazione e azione.

#### Trasformazione

Sul RDD di input è possibile imporre delle trasformazioni per ottenere in output un RDD differente dal primo.

Le trasformazioni basilari utilizzabili sono descritte dai seguenti metodi:

- Map: permette dato un RDD di ingresso di trasformarne il contenuto che vi è all'interno. Il numero di dati in ingresso e in uscita rimane uguale;

- FlatMap: concettualmente le possibili trasformazioni operate sono le medesime della map, con la differenza che queste ritornano una collezione di dati. Il numero di dati in input è minore rispetto ai dati in output;

- Filter: permette di imporre sui dati una condizione booleana al fine di generare un filtro. Il numero di dati risultante può essere minore rispetto al numero di dati in input.

### Azione

Dall'RDD di input si ottiene come output un risultato locale nella nostra applicazione (ad esempio un variabile di tipo intero, una collezione di dati, ...). Le azioni basilari sono:

- Count: restituisce il numero di dati presente nel RDD;
- Collect: permette di ritornare una lista di dati presenti all'interno del RDD;
- Reduce: impone una determinata condizione tra due istanze di dati per poter restituire una variabile che descriva l'operazione utilizzata. Un possibile esempio di questa metodologia permette di ottenere un valore massimo di un campo all'interno dei dati.

### Paired RDD

Esiste una versione di tipo key-value del RDD [11], costituito da due campi:

- Key: è la chiave di un'istanza, utile per effettuare delle operazioni sui dati che hanno in comune la stessa chiave;
- Value: è la parte effettiva in cui vengono memorizzati i dati, attribuiti ad una relativa chiave. Dove sono disponibili un set di metodi simili alla versione normale del RDD.

### Direct Acyclic Graph (DAG)

Il DAG [10] è un grafo che descrive la sequenza di operazioni, di tipo trasformazione e azione, all'interno della nostra applicazione.

In questa tesi, verrà usata questa tipologia di rappresentazione per descrivere le operazioni svolte al fine di migliorarne la comprensione.

Il DAG è composto da due elementi:

- Nodo: definisce la trasformazione o l'azione che avviene sul RDD;
- Arco mono direzionale: indica la prossima operazione che si dovrà effettuare.

### DataFrame

In Spark, c'è la possibilità di organizzare i dati con un altro tipo di oggetto chiamato DataFrame [12], che consiste in una struttura a forma di colonne, simile alle tabelle dei database relazionali. Quando si crea un nuovo DataFrame bisogna definire lo schema delle colonne che in nostri dati dovranno assumere.

I vantaggi derivanti dai DataFrame sono:

- Dare una forma strutturata ai nostri dati in modo semplice senza dover creare classi customizzate che descrivano i nostri dati;
- Utilizzare metodi che richiamano molto l'approccio utilizzato nel mondo SQL;
- Introduzione di un ottimizzatore per l'esecuzione delle "query" svolte.

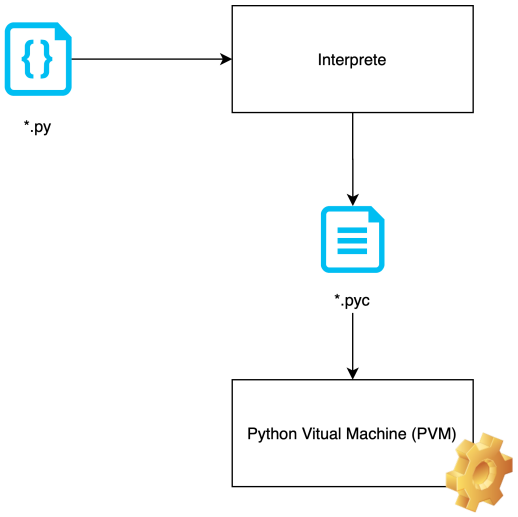


*Figura 2.7: Modello di esecuzione di uno script Python*

### 2.9.2 Python

Per sviluppare questa tesi è stato utilizzato Python [14] [2] come linguaggio di programmazione orientato agli oggetti. È un linguaggio interpretato, per questo motivo il codice non viene compilato e non si ottiene di conseguenza l'eseguibile del programma. Nell'eseguire gli script su un calcolatore, è necessario avere un interprete Python che permetta di convertire il codice in un formato di livello intermedio chiamato bytecode. L'interprete effettua i seguenti passi:

- Verifica della correttezza sintattica dello script;
- Traduce lo script in bytecode.

Una volta ottenuto il bytecode, ne segue l'elaborazione dalla Python Virtual Machine (PVM) che, a sua volta, esegue lo script di partenza (figura 2.7).

Python, confronto ad altri linguaggi di programmazione, ha delle performance peggiori, poiché il compilatore non genera l'eseguibile come avviene in c/c++, ma il codice viene solo interpretato. Il vantaggio di Python è l'essere molto semplice da utilizzare e da imparare, poiché nasce già con delle astrazioni utili per semplificarne l'utilizzo allo sviluppatore.

Per sfruttare le funzionalità di Spark con Python è necessario utilizzare la libreria PySpark. Un'altra motivazione riguardante la scelta di PySpark come libreria di sviluppo della tesi, deriva dal fatto che è molto utilizzata nella community di sviluppatori, così, nel caso di necessità, si possono adoperare soluzioni a problemi ricorrenti reperibili in rete.

Inoltre, sono state utilizzate altre due librerie:

- Scikit-learn [15]: libreria che permette di fare task di machine learning;
- Matplotlib [13]: libreria per andare a creare dei grafici utili per presentare i dati.

### 2.9.3 Software utilizzati

I software utilizzati nello sviluppo di questa tesi sono:

- JupiterLab: ambiente di sviluppo che permette di creare/eseguire notebook e script scritti in Python, con possibilità di suddividere il codice in blocchi e di definire dei blocchi testuali con la sintassi del Markdown.
- PyCharm: IDE sviluppato da JetBrains, che permette sviluppare codice Python in locale e di debbugare in modo semplice il codice scritto.

# Capitolo 3

# Sviluppo del lavoro

All'interno di questo capitolo si descriveranno le metodologie che vi sono dietro allo sviluppo di questa tesi e l'utilizzo del classificatore associativo per le predizioni.

## 3.1 Descrizione del problema

Per poter classificare situazioni particolari in base ai concetti di tempo e spazio, bisogna partire dal dataset di studio e manipolarlo, cercando di trasformare le informazioni in un formato che ci permetta di facilitare l'analisi. Il fine ultimo della trasformazione è quello di generare delle sequenze temporali, che descrivano al loro interno l'evoluzione temporale delle stazioni.

Il problema che scaturisce nell'analizzare il nostro dataset di ingresso è influenzato dai seguenti fattori:

- Record non continui in base al timestamp;
- Valori di alcuni campi non coerenti tra di loro con il passare del tempo.

Questi due fattori portano ad effettuare il pre-processing per il cleaning dei dati.

Come si può vedere dal dataset, è presente un campo che definisce il concetto di tempo mediante un timestemp, ossia una combinazione della rappresentazione della data e dell'orario in cui i dati vengono campionati.

Nella manipolazione dei dati, bisogna introdurre un parametro di granularità, che permette di modificare l'orario e la data di un dato record, in modo tale da attribuire alle situazioni di tempo un multiplo della granularità, così da suddividere in istanti temporali le situazioni che vi sono nelle stazioni.

Questo è il primo passo che si deve compiere per poter rendere possibile la creazione delle sequenze. Una volta fatto questo, bisognerà passare al raggruppamento di questi istanti di tempo con le diverse stazioni associate ad essi e i relativi stati.

Le sequenze temporali sono composte dall'insieme delle finestre consecutive all'istante di tempo iniziale, cioè quello che si sta valutando. Bisogna, inoltre, fare attenzione al fatto che il primo istante della sequenza da generare sia incluso nel dataset di partenza e in seguito validare le finestre temporali affinchè ogni istante sia successivo al precedente. Si noti anche che una finestra temporale può essere presente all'interno di più sequenze, ma in posizione diversa.

I possibili stati che una stazione può assumere sono:

- Full: quando una stazione ha tutti gli slot delle biciclette occupate;

- Empty: quando gli slot delle biciclette sono tutti vuoti;
- AlmostFull: quando gli slot sono quasi tutti pieni;
- AlmostEmpty: quando gli slot sono quasi tutti vuoti;
- Normal: quando la stazione non si trova in nessun caso tra i precedenti.

Il nostro obiettivo è quello di riuscire a capire lo stato delle stazioni e distinguere tra stato di Full, Empty e Normal. Questo è il motivo per cui si vuole utilizzare un algoritmo basato sull'estrazione di regole di associazione per poter trovare situazioni di alta rilevanza che ci permettano di capire in quale stato si trovi una stazione in base alle situazioni pregresse. Come si può notare, per generare lo stato di AlmostFull e di AlmostEmpty è necessario un parametro di threashold, un parametro da studiare in modo consono per capire quando attribuire alle stazioni queste tipologie di stati.

## 3.2 Dataset

In questa analisi, il dataset considerato è contenuto all'interno di un file csv, con la seguente struttura:

$$Timestamp, StationID, used, free$$

Ogni riga del file è caratterizzata dall'identificativo della stazione (StationId) e il timestamp che corrisponde all'istante di tempo in cui è stato generato il record; il seguente formato:

$$YYYY - MM - DD\ HH : MM : SS$$

Le seguenti lettere rappresentano:

- YYYY: l'anno;
- MM: Mese;
- DD: giorno del mese;
- HH: ora;
- MM: minuti;
- SS: secondi.

Il formato utilizzato della data è quello USA, poiché molto utile per effettuare i confronti tra i timestamp. Infine, i valori di used e i valori di free definiscono rispettivamente il numero di biciclette utilizzate e il numero di biciclette disponibili in quella stazione in quel preciso istante di tempo.

Di seguito viene riportato un esempio (3.2.1) di possibili record all'interno del file.

**Esempio 3.2.1.** Dataset in input.

2008-05-15 12:00:00,1,18,0
2008-05-15 12:10:00,1,18,0
2008-05-15 12:15:00,1,0,18
2008-05-15 12:05:00,2,0,21
2008-05-15 12:11:00,2,0,21
2008-05-15 12:18:00,2,0,21
2008-05-15 12:01:00,3,24,0

## 3.3 Trasformazione del dataset

### 3.3.1 Descrizione dell'implementazione

L'implementazione della trasformazione del dataset, da cui si estrarranno le regole di associazione, è basata sulla classe GenerateDataset. Questa classe ha una relazione 1 a 1 con la classe Configuration, la quale raggruppa i parametri di granularità temporale (granularity) e la grandezza delle sequenze temporali (window).

La classe GenereteDataset permette la trasformazione dei dati e la suddivisione del dataset in training e test set con i seguenti metodi:

- generete: metodo in cui avviene l'adattamento del dataset;
- generate_sequence: metodo che si occupa della creazione delle sequenze temporali;
- generate_trainAndTest: metodo che si occupa della creazione del training e test set, dove le stazioni possono assumere i seguenti stati:
  - Full;
  - Empty;
  - Normal;
  - AlmostFull;
  - AlmostEmpty.
- generate_trainAndTest_normal: simile al metodo precedente, vengono estratte stazioni aventi solamente gli stati:
  - Full;
  - Empty;
  - Normal.

Nella figura 3.1 è possibile visionare il diagramma UML relativo a queste due classi descritte in precedenza.

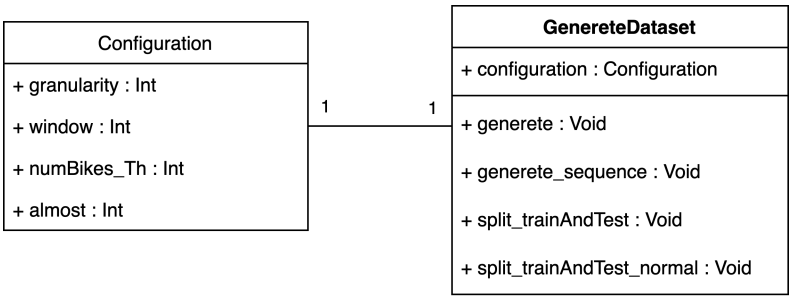


*Figura 3.1: UML delle classi Configuration e GenerateDataset*

## 3.4 Preparazione del formato di ingresso

La prima manipolazione fatta sul nostro dataset è il filtraggio della prima riga che costituisce l'header del file, con l'utilizzo del metodo filter disponibile nell'oggetto RDD.

Ad ogni record bisogna attribuire lo stato in qui la stazione si trova, in un relativo istante di tempo. Tale stato può essere dedotto dalla situazione presente nei valori di used e free; i possibili casi sono:

- Se free è uguale a 0: la stazione in quell'istante di tempo è sprovvista di biciclette, la situazione è definita come vuota (Full);
- Se used è uguale a 0: la stazione in quell'istante di tempo è completa di biciclette, la situazione è definita come piena (Empty);
- Se il valore di free è minore o uguale al valore di soglia presente nella classe Configuartion, la stazione è nello stato di quasi piena (AlmostFull);
- Se il valore di used è minore o uguale al valore di soglia presente nella classe Configuartion, la stazione è nello stato di quasi vuota (AlmostEmpty);
- Se non vale nessuno dei precedenti, lo stato della stazione è normale (Normal).

L'attribuzione dei vari stati viene svolta attraverso l'utilizzo del metodo map, che richiama dentro di sé una funzione di nome assignStatus. Di conseguenza viene riportato un esempio (3.4.1) che si genera quando viene eseguito il metodo map.

**Esempio 3.4.1.** Introduzione dello stato per ogni record.

2008-05-15 12:00:00,1,full
2008-05-15 12:05:00,1,full
2008-05-15 12:10:00,1,full
2008-05-15 12:15:00,1,empty
2008-05-15 12:05:00,2,empty
2008-05-15 12:11:00,2,empty
2008-05-15 12:18:00,2,empty
2008-05-15 12:01:00,3,full

Una volta avute le stazioni con i relativi stati di apparenza, bisogna effettuare una trasformazione sull'allineamento dei timestamp in accordo con il parametro di granularità.

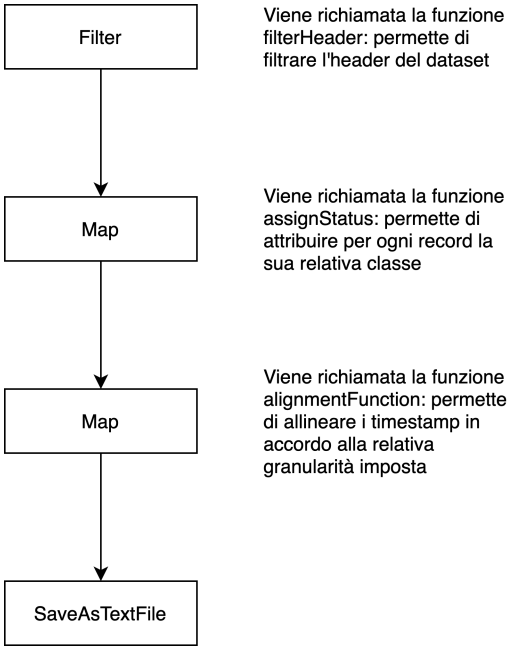


*Figura 3.2: DAG del metodo generate*

Per allineamento si intende la trasformazione dei minuti (eventualmente le ore o anche la data) del timestamp nel minuto più basso rispetto alla granularità. Il risultato finale è nell'avere il timestamp multiplo della granularità, come si può vedere nell'esempio 3.4.2.

**Esempio 3.4.2.** Manipolazione con granularity pari a 5.
2008-05-15 12:00:00,1,full
2008-05-15 12:00:00,3,full
2008-05-15 12:05:00,1,full
2008-05-15 12:05:00,2,empty
2008-05-15 12:10:00,1,full
2008-05-15 12:10:00,2,empty
2008-05-15 12:15:00,1,empty
2008-05-15 12:15:00,2,empty

Questa tipologia di trasformazione ci aiuterà nei prossimi passi ad affrontare il problema della generazione delle sequenze temporali.

Nella figura 3.2 è riportato il DAG delle operazioni che sono all'interno del metodo generete.

### Trasformazione in finestre temporali e creazione delle sequenze temporali

A questo punto, si vogliono ottenere come risultato in output le sequenze temporali, utilizzate dall'algoritmo di FP-Growth per la generazione di regole.

Una sequenza temporale del nostro dataset è definita nel seguente modo:

$$0\_\{StationID - State, \ldots\} \rightarrow 1\_\{StationID - State, \ldots\} \rightarrow$$

$$\ldots \rightarrow (window - 1)\_\{StationID - state, \ldots\}$$

Dove:

- StationID: è l'identificativo della stazione;
- State: è lo stato della stazione in quel delta temporale;
- Window: è la grandezza della nostra sequenza temporale.

All'interno di ogni finestra temporale, definito da un delta, possiamo trovare una lista di stazioni con il relativo stato.

Ogni sequenza temporale deve iniziare obbligatoriamente con un valore di delta pari a 0 e ogni delta successivo deve essere il consecutivo della finestra precedente, si ottiene in output una stringa avente più finestre temporali.

È da notare che nell'ultima finestra all'interno della sequenza temporale, non è detto che il delta assuma valore pari a window-1, ma può assumere anche valori minori.

Di seguito viene riportato un esempio 3.4.3 di sequenze temporali in accordo con gli esempi precedenti.

**Esempio 3.4.3.** Costruzione sequenza con lunghezza pari a 3 (w = 3).
$0\_\{1\text{-}full, 3\text{-}full\} \rightarrow 1\_\{1\text{-}full, 2\text{-}empty\} \rightarrow 2\_\{1\text{-}full, 2\text{-}empty\}$
$0\_\{1\text{-}full, 2\text{-}empty\} \rightarrow 1\_\{1\text{-}full, 2\text{-}empty\} \rightarrow 2\_\{1\text{-}empty, 2\text{-}empty\}$
$0\_\{1\text{-}full, 2\text{-}empty\} \rightarrow 1\_\{1\text{-}empty, 2\text{-}empty\}$
$0\_\{1\text{-}empty, 2\text{-}empty\}$

Per la generazione delle sequenze temporali è necessario definire tutte le finestre possibili.

Come prima cosa, bisognerà selezionare il valore minimo di timestamp presente nei record, questa operazione può essere fatta utilizzando il metodo reduce sul nostro RDD richiamando la funzione selectMinTimestamp, che compie confronti locali tra le due stringhe in ingresso. Il risultato dall'azione di reduce è una stringa rappresentante il timestamp minimo, utilizzato per non generare finestre temporali inferiore a questo valore.

Di conseguenza, per ogni record possiamo definire, in accordo con suo timestamp, le finestre temporali a cui dovrà appartenere. Il metodo flatMap, ci permette di generare una lista che verrà inserita all'interno del nostro RDD di output partendo da un record all'interno a quello di input.

All'interno della flatMap viene richiamata una funzione di nome definitionOfDelta, utilizzando il record di input verrà creata una lista, avente delle entry caratterizzate da una struttura key-value. Rispettivamente il valore di key-value assumerà il seguente significato:

- Key: corrisponde al delta temporale del record di cui farà parte, in questo caso definito dal timestamp riadattato. La chiave è costituita da:

$$timestamp\_delta$$

- Value: corrisponde alla stringa definita nel seguente modo:

$$StationID - state$$

La generazione del timestamp riadattato permette al campo value di entrare nelle finestre temporali (precedenti o correnti in base al timestamp di partenza) definite dal delta rappresentato nel campo chiave. Nell'implementazione, la generazione del delta tiene conto dei cambi di giorno e di anno, poiché dato un record si genera, possibilmente, una lista di lunghezza pari a valore di window. Si tiene conto anche del minimo valore di timestamp pre-calcolato in precedenza, per non generare dei delta temporali che non sono compresi nel dataset, poiché sarebbe controproducente.

Per rendere migliore la comprensione dell'output di questa funzione, viene riportato un esempio pratico (3.4.4).

**Esempio 3.4.4.** Il seguente dato di input nella funzione definitionOfDelta:
2008-05-15 12:15:00,2,empty
L'output sarà:
(2008-05-15 12:15:00_0, 2-empty)
(2008-05-15 12:10:00_1, 2-empty)
(2008-05-15 12:05:00_2, 2-empty)

Il passo successivo permette di ridurre i record, utilizzando il metodo reduceByKey, in base al valore presente nella key. Il metodo reduceByKey richiama la funzione reducePartialSequence, permette di unire i campi value di due record utilizzando un carattere speciale definito da una virgola. All'interno di questo metodo si introduce anche un ordimento alfanumerico dei valori, come si può evincere dall'esempio 3.4.5.

**Esempio 3.4.5.** Il seguente dato di input nella funzione reducePartialSequence:
("2008-05-15 12:15:00_0", "1-empty")
("2008-05-15 12:15:00_0", "2-empty")
L'output sarà:
("2008-05-15 12:15:00_0", "1-empty,2-empty")

Nell'esempio precedente (3.4.5) nel campo value abbiamo una stringa contenete le stazioni con i relativi stati. Si nota che non vi sono duplicazioni degli StationId.

Il passo successivo è di incorporare nel timestamp di partenza le stringhe che definiscono la finestra temporale. Tuttavia, è necessario riadattare i valori di key-value:

- Key: assume semplicemente il valore di timestamp di partenza della sequenza:

$$timestamp$$

- Value: assume una combinazione con il delta relativo al record e al value precedente:

$$delta\_\{StationId - State, ...\}$$

La procedura da utilizzare al fine di raggiungere questo formato è quella di adattare i record, richiamando il metodo map sul RDD. All'interno della map si effettua una chiamata alla funzione trasform, nell'esempio 3.4.6 è riportata una sua applicazione.

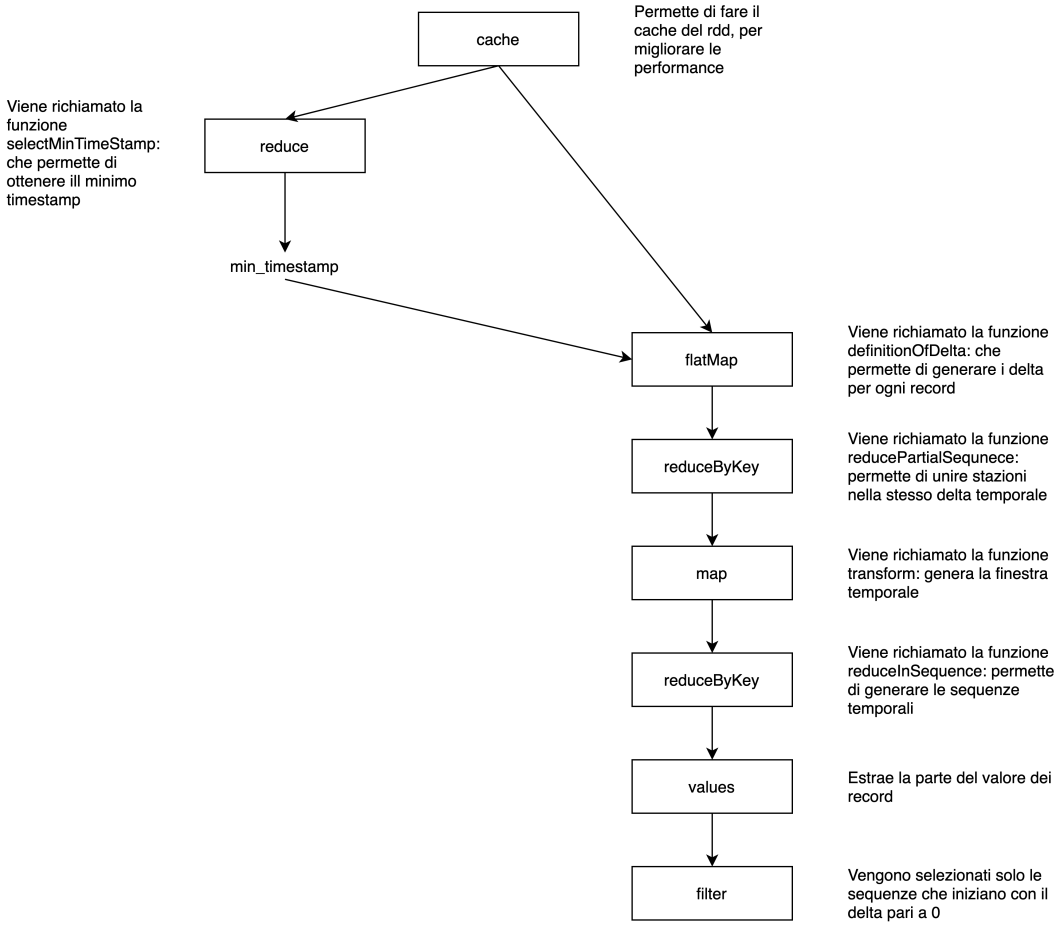


*Figura 3.3: DAG del metodo generete_sequence*

**Esempio 3.4.6.** Dato il seguente dato di input nella funzione trasform:
("2008-05-15 12:15:00_0", "1-empty,2-empty")
L'ouput sarà:
("2008-05-15 12:15:00", "0_{1-empty,2-empty}")

La generazione delle sequenze viene operata grazie all'utilizzo di una reduceByKey, permette di collegare in modo ordinato le finestre temporali di interesse associate al valore di timestamp di inizio definito nel campo key.

Si possono presentare delle sequenze aventi come primo delta temporale diverso da 0. Queste situazioni devono essere rimosse utilizzando il metodo filter, che va a testare se la sequenza inizia con 0.

Nella figura 3.3, viene riportato il DAG del motodo generete_sequence, che si occupa della creazione delle sequenze temporali.

### 3.4.1 Generazione del training e test

La generazione del training e test set avviene in modo statico, grazie all'utilizzo del metodo generate_trainAndTest, che riceve in input la grandezza che il test set deve assumere. La grandezza di default del training set è pari a $\frac{3}{4}$, invece per il test set è pari a $\frac{1}{4}$.

La divisione è effettuata semplicemente utilizzando il metodo zipWithIndex, disponibile all'interno dell'oggetto RDD, che permette di enumerare tutti i record in modo crescente. Il formato risultante sarà:

$$(sequenza\ temporale, index)$$

La selezione dei dati, che comporranno il training set, sarà determinata grazie all'utilizzo del metodo filter, che imporrà il seguente vincolo:

$$index < (1 - test\_size) \cdot dataset\_size$$

dove:

- Index: è l'indice relativo al record che si sta valutando in quel momento;
- Test_size: grandezza presa in input da questo metodo;
- Dataset_size: numero di sequenze temporali.

Per costituire il test set verranno svolte le medesime operazioni, utilizzando un altro tipo di condizione:

$$index \geq (test\_size) \cdot dataset\_size$$

#### Manipolazione dello stato Almost e Normal

Il training set deve essere manipolato per poter avere le seguenti proprietà:

- Le sequenze aventi le stazioni nello stato Normal possono essere rimosse dalla sequenza, per poter rendere i confronti durante la fase di test più veloci, poiché la non presenza di una stazione implica lo stato Normal;
- Le situazioni di Full, Empty, AlmostFull o di AlmostEmpty vengono mantenute invariate.

La manipolazione del test set, invece, avverrà nel seguente modo:

- Si applicano le stesse manipolazioni che vengono fatte sul training solo fino alla penultima finestra temporale;
- Per l'ultima finestra temporale si applica la trasformazione dello stato AlmostFull e AlmostEmpty nello stato Normal, per effettuarne la predizione. In questo caso le stazioni nello stato Normal non verranno rimosse, poiché si effettuare la predizione su quelle stazioni.

Questo adattamento delle sequenze viene fatto dal metodo map che richiama la funzione removeStationInNormalState, che prende in input un valore booleano che indica se si sta trattando una sequenza di training oppure di test.

Nell'esempio 3.4.7 si può evincere come avviene la suddivisione del training e del test set.

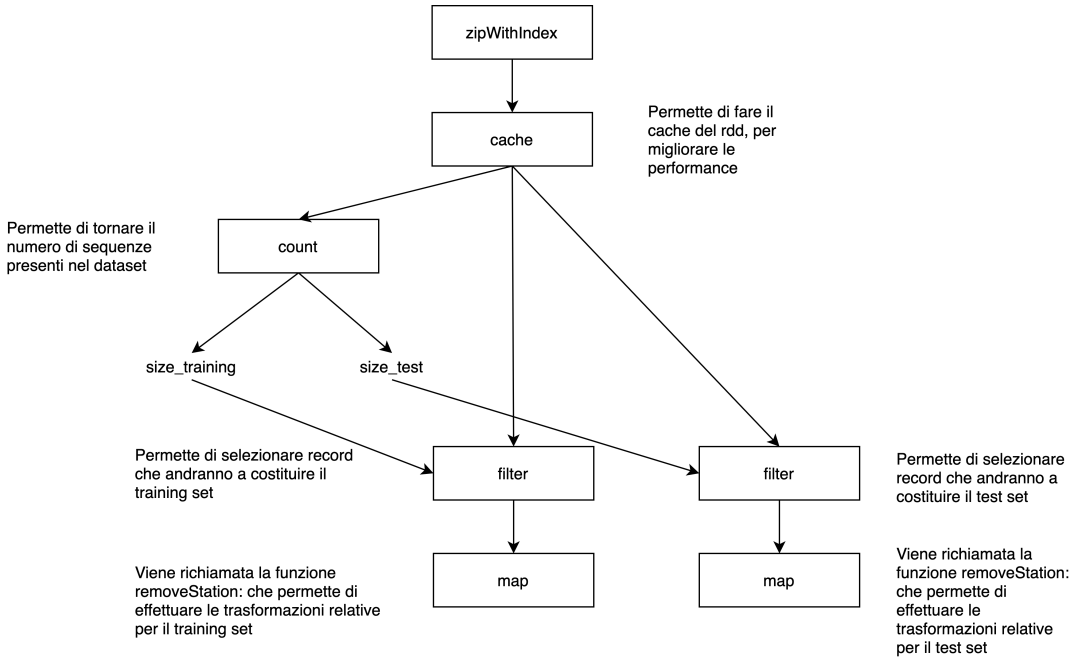


*Figura 3.4: DAG del metodo generete_generate_trainAndTest.*

**Esempio 3.4.7.** Generazione del training e test set, partendo dai seguenti dati:
$0\_\{1-full, 3-full\} \rightarrow 1\_\{1-full, 2-empty\} \rightarrow 2\_\{1-full, 2-empty\}$
$0\_\{1-full, 2-empty\} \rightarrow 1\_\{1-full, 2-empty\} \rightarrow 2\_\{1-empty, 2-empty\}$
$0\_\{1-full, 2-empty\} \rightarrow 1\_\{1-empty, 2-empty\}$
$0\_\{1-empty, 2-empty\}$
Output del metodo zip:
$(0\_\{1-full, 3-full\} \rightarrow 1\_\{1-full, 2-empty\} \rightarrow 2\_\{station1-full, station2-empty\}, 0)$
$(0\_\{1-full, 2-empty\} \rightarrow 1\_\{1-full, 2-empty\} \rightarrow 2\_\{1-empty, 2-empty\}, 1)$
$(0\_\{1-full, 2-empty\} \rightarrow 1\_\{1-empty, 2-empty\}, 2)$
$(0\_\{1-empty, 2-empty\}, 3)$
Numero dei sequenze è pari a 4 e supponendo di impostare la grandezza del test set pari a 1/3 del dataset otteniamo:

- Training:

  $(0\_\{1-full, 3-full\} \rightarrow 1\_\{1-full, 2-empty\} \rightarrow 2\_\{1-full, 2-empty\}, 0)$

  $(0\_\{1-full, 2-empty\} \rightarrow 1\_\{1-full, 2-empty\}) \rightarrow 2\_\{1-empty, 2-empty\}, 1)$

- Test:

  $(0\_\{1-full, 2-empty\} \rightarrow 1\_\{1-empty, 2-empty\}, 2)$

Il seguente record verrà scartato perché ha solo una finestra temporale:
$(0\_\{1-empty, 2-empty\}, 3)$

Per semplificare la comprensione viene riportato il DAG delle operazioni svolte (figura 3.4).

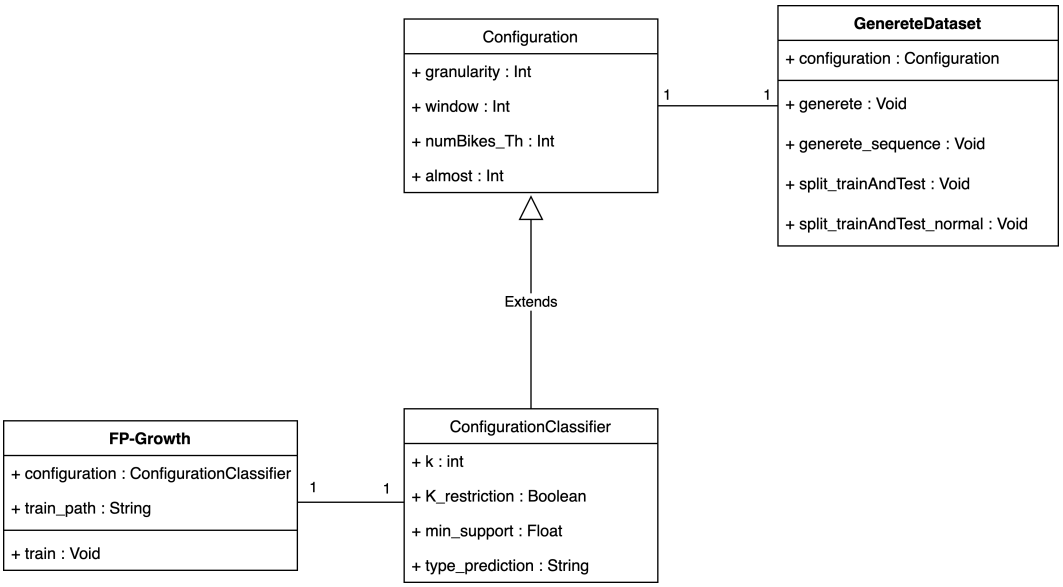


*Figura 3.5: UML delle classi: FP-Growth, Configuration, ConfigurationClassifier e GenerateDataset.*

Al fine di generare i dati di training e di test con solamente gli stati di Full, Empty e Normal, si utilizza il metodo generate_trainAndTest_normal, che permette di riaddattare la suddivisione svolta in precedenza. Le operazioni svolte sono le stesse del metodo genererate_trainAndTest, ma utilizzando la funzione removeStationInAlmostState nell'operazione di map.

## 3.5 Estrazione delle regole di associazione

L'estrazione delle regole di associazione è il punto saliente del nostro processo, poiché la classificazione si basa su di esse, come si può evincere dal capitolo precedente 2.5.

Per l'estrazione delle regole di associazione si utilizzerà l'algoritmo di FP-Growth, implementato all'interno della libreria di MLIb di Spark [8]. Utilizzeremo questa implementazione come se fosse una scatola chiusa vedendo solamente i dati generati in output.

I dati generati in output dovranno essere manipolati in modo consono alla nostra analisi.

### 3.5.1 Implementazione della estrazione di regole

La classe FP-Growth permette di ricevere nel suo costruttore un oggetto di tipo ConfigurationClassifier, al cui interno sono definiti i parametri utili per la creazione del FP-Tree e il path relativo dal training set.

La classe ConfigurationClassifier deriva dalla classe Configuration, in questo caso specifico il parametro utile per la generazione di regole è il valore di minimo supporto. Nella figura 3.5 è possibile vedere la relazioni risultanti dalle classi descritte fino ad ora.

In questa classe l'implementazione del FP-Growth è quella derivata da PySpark. La creazione della classe custom permette di inglobare l'implementazione definita in PySpark per rendere il codice più comprensibile.

| antecedent | consequent | conf. | lift |
|---|---|---|---|
| [0_{1-full, 3-full, 1_{1-full, 2-empty}] | [2_{1-full, 2-empty}] | 1.0 | 2.0 |
| [0_{1-full, 2-empty] | [1_{1-full, 2-empty}] | 1.0 | 1.0 |
| [0_{1-full, 2-empty}, 1_{1-full, 2-empty}] | [2_{1-empty, 2-empty}] | 1.0 | 2.0 |
| [0_{1-full, 3-full}] | [1_{1-full, 2-empty}] | 1.0 | 1.0 |

*Tabella 3.1: DataFrame ritornato dal metodo fit della classe FP-Growth.*

All'interno della nostra classe vi è un metodo di nome train che permette la creazione delle regole di associazione. Al suo interno vengono letti dal HDFS le sequenze di training trasformate in DataFrame, la colonna "items" descrive una lista di tutte le finestre temporali che costituiscono la sequenze temporale, come si può notare dall'esempio 3.5.1.

**Esempio 3.5.1.** Per la generazione del DataFrame, si può partire dal seguente dato di training:
$0\_\{1-full, 3-full\} \rightarrow 1\_\{1-full, 2-empty\} \rightarrow 2\_\{1-full, 2-empty\}$
e si ottiene il seguente Dataframe generato, con unica colonna di nome "Items":
$Items = [0\_\{1-full, 3-full\}, 1\_\{1-full, 2-empty\}, 2\_\{1-full, 2-empty\}]$

Prima di creare l'oggetto FP-growth di PySpark, è necessario compiere un controllo sul supporto minimo imposto, definito dal seguente vincolo:

$$trainingSize \cdot min\_support \geq 1$$

Se non viene rispettato, lo script effettua un'uscita dall'esecuzione utilizzando una exit presente nella liberia sys di Python.

Una volta creato l'oggetto di FP-Growth di PySpark e eseguito il metodo fit passandogli il DataFrame creato in precedenza, si otterà un DataFrame avente la seguente struttura:

$$antecedent, consequent, confidence, lift$$

Il DataFrame risultante dall'esecuzione del metodo fit della classe FP-Growth è riportato nella tabella 3.1.

Nel caso in cui si stia facendo l'analisi con le sequenze aventi le stazioni nello stato AlmostFull o AlmostEmpty, bisogna effettuare una manipolazione della regola che nel conseguente non abbia stazioni nello stato di AlmostFull e AlmostEmpty, poiché il loro stato reale è Normal. Per fare questo bisogna utilizzare una User Defined Function (UDF) che permette di manipolare il DataFrame per avere questo tipo di modifica. Questa operazione viene svolta della funzione filterAlmostInConseguent, nella quale semplicemente sul campo conseguent della nostra regola si rimuovono tutte le occorrenze delle stazioni che sono nello stato AlmostFull o AlmostEmpty, poiché la non presenza di una stazione ne definisce lo stato come Normal.

Il prossimo passo è quello di filtrare le regole, utilizzando la UDF di nome filterForAR, affinché soddisfino:

- Ogni delta temporale presente nell'antecedete sia minore del delta presente nel conseguente;

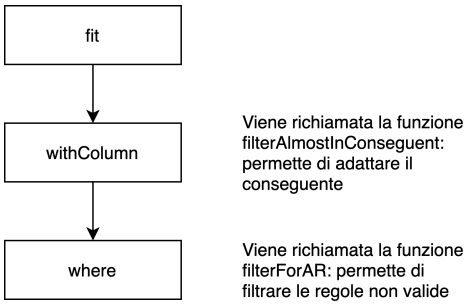


*Figura 3.6: DAG del metodo train della classe FP-Growth.*

- Ci sia il delta pari a zero all'interno dell'antecedente.

Se questi due vincoli non vengono soddisfatti, la regola viene scartata. Per semplicità viene riportato il DAG descritto dal metodo train nella figura 3.6.

## 3.6 Classificazione

La classificazione si basa sulle regole di associazione estratte mediante l'utilizzo della classe FP-Growth. In questo capitolo verranno descritte le metodologie per effettuare la classificazione dei dati di test, prevedendo lo stato delle stazioni presenti nell'ultima finestra temporale della sequenza.

### 3.6.1 Introduzione alla classificazione

La classificazione può avvenire in vari modi in base al tipo di configurazione che si è scelta sul classificatore. La selezione del tipo di configurazione che il nostro modello dovrà adottare sarà oggetto di un'analisi approfondita nel capitolo riguardante la sezione sperimentale.

La parte della classificazione può partire quando le regole di associazione sono state selezionate.

Per compiere la classificazione sulle regole di associazione si dovrà prendere la stazione presente nell'ultima finestra temporale nel record di test e cercare le regole che soddisfino le situazioni precedenti delle finestre temporali, considerando il corpo della regola estratta dai dati di training.

Da un dato di test può essere selezionata più di una regola, queste regole possono essere utilizzate da più criteri per poter compiere la predizione. Per definire la predizione di una stazione, data la regola selezionata, basta accedere alla testa della regola prendendo lo stato che è attribuito alla stazione considerata.

Le tipologie di classificazione che si possono avere sono date dai tipi di training e di test set che si sono generati. Possiamo definire due casi:

- Training e test set con solo gli stati di Full, Empty e Normal;
- Training e test set con gli stati di Full, Empty, AlmostFull, AlmostEmpty e Normal, per poter aiutare il nostro modello a riconoscere meglio i vari stati.

In base alle regole selezionate da un relativo dato di test, possiamo definire più modalità di classificazione. Il concetto alla base è quello che si possono avere due modalità di classificazione di un record, infatti si può:

- Basarsi solo sulla prima regola di associazione che si trova;
- Basarsi su più regole e andare ad effettuare un voto di maggioranza o un voto pesato in base all'importanze delle regole selezionate.

Queste modalità sono descritte dai parametri di configurazione del nostro modello e anche essi dovranno essere selezionati a dovere.

Un'altra implementazione che si può introdurre è quella basata sulle fasce orarie che caratterizzano in modo divisivo le predizioni di un dato timestamp, ad esempio in che periodo del giorno si trova. Questo classificatore sarà un caso derivato dai casi descritti in precedenza, poiché si potrà sempre effettuare un'analisi per stabilire quale tipologia di training e test set selezionare. In questo caso deve essere introdotto un altro parametro, che descrive le fasce temporali utilizzate dal nostro classificatore per creare diversi insiemi di regole di associazione su un determinato valore di fascia temporale, così da poter effettuare il training solo sulle sequenze che ricadono in quella fascia oraria. Al test time viene selezionato il set di regole di associazione relativo al timestamp di inizio della sequenza temporale del record di test.

### 3.6.2 Linea guida sulla classificazione

Dato un record del test set, per poter predire quale stato attribuire ad una data stazione, bisogna prendere tutte le regole estratte del training set che soddisfino dei requisiti:

1. Si tengono in considerazione soltanto le regole che hanno la stazione presa in esame presente nel conseguente della regola;
2. Le regole selezionate devono avere un match con il record preso. Per match si intende che il conseguente contenga le finestre temporali presenti nel record di test (a parte l'ultima finestra temporale);
3. Se nessuno dei due casi viene soddisfatto, la stazione viene classificata nello stato Normal.

Se nel secondo punto vengono selezionate più di una regola bisogna poter adottare diverse strategie. Il primo passo, prima di poter definire queste strategie, è quello di ordinare le regole selezionate in corrispondenza dei seguenti campi:

- Confidenza;
- Lift;
- Lunghezza della regola: si prende quella più lunga;
- Ordine lessico grafico (bisogna riordinare i delta facendo una map).

Questo procedimento ci permette di ottenere le regole ordinate in base alla loro importanza. Per scegliere quale stato attribuire alla corrispondente stazione possiamo chiederci come poter utilizzare queste regole ordinate e possiamo definire i seguenti casi:

- Definire con un parametro K quante regole selezionare che diano il loro contributo nella classificazione:

  - Se K è uguale a 1, si prende la prima regola presente nella partizione e lo stato presente in essa, dopo si associa quello stato alla stazione considerata;
  - Se K è maggiore di 1, possiamo definire delle metodologie per attribuire lo stato:
    * Effettuare un voto di maggioranza: lo stato più frequente viene scelto come predizione;
    * Effettuare una selezione dello stato associato ad uno score che viene creato in modo pesato in accordo con l'importanza delle regole che hanno quella stazione in un determinato stato.

- Inoltre, c'è la possibilità di definire un altro vincolo relativo al numero di regole selezionate e questo viene definito attraverso un parametro booleano K_restriction:
  - Se ha valore "true" e se il numero di regole selezionate non è pari a k, allora la stazione relativa viene classificata nello stato Normal;
  - Se ha valore "false" e se il numero di regole selezionate non è pari a k, allora si effettua la classificazione in modo normale in base agli altri parametri passati.

### 3.6.3 Possibili soluzioni

Le prime soluzioni che si possono adottare per compiere la classificazione sono:

- Ciclare su le regole di associazione: come prima cosa si vanno ad estrarre le regole di associazione e si salvano in memoria principale utilizzando il metodo collect sul RDD. Una volta lette dall'HDFS i dati di test, si può operare su ogni dato di testing, utilizzando il metodo map, un'iterazione su ogni regola di associazione in memoria principale così ad effettuare la classificazione, prendendo in considerazione le regole che hanno più rilevanza;

- Ciclare sui dati di test: un'altra possibilità ma molto più complicata da realizzare è quella di salvarsi in memoria principale i dati di testing e di ciclarli all'interno di ogni regola di associazione presente nel RDD così da effettuare una predizione su di esse. Bisogna tenere conto che ogni predizione viene fatta da ogni regola di associazione presente nel RDD, effettuando un filtraggio ulteriore per capire quali di esse hanno maggiore rilevanza.

Come si può notare queste due soluzioni non sono ideali quando bisogna trattare una mole elevata di dati per via dell'iterazione per compiere una predizione di un dato di test.

Bisognerebbe evitare l'operazione di collect del contenuto presente all'interno del RDD, poiché i dati non potrebbero stare all'interno della memoria principale del Cluster Manager.

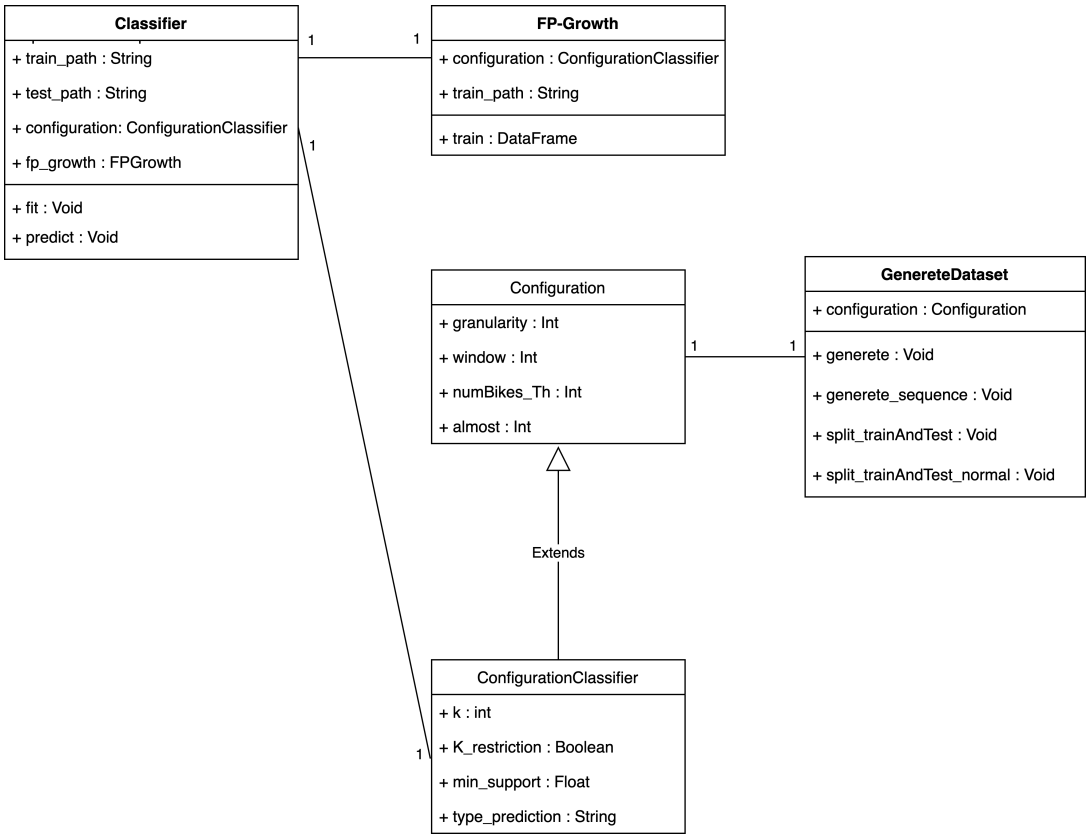


*Figura 3.7: UML delle classi: Classifier, FP-Growth, Configuration, ConfigurationClassifier e GenerateDataset.*

### 3.6.4 Soluzione proposta

Si basa sull'utilizzo esclusivo di trasformazioni e azioni sull'RDD. I tipi di trasformazione utilizzati, sono derivati dai metodi presenti in SparkSQL, che utilizzano costrutti derivati dal mondo SQL per manipolare in modo semplice i nostri record. La soluzione proposta è contenuta all'interno della classe Classifier, che dispone del metodo fit per creare le regole di associazione sulle sequenze temporali di training. Il metodo predict, invece, permette di compiere la predizione delle stazioni contenute all'interno delle sequenze di test.

Nella figura 3.7 è possibile visionare come la classe Classifier si relaziona con le altre classi presentate in precedenza.

Nei seguenti esempi si farà riferimento ai seguenti training e test set definiti in questo modo:

$train = [0\_\{1 - full, 3 - full\} \rightarrow 1\_\{1 - full, 2 - empty\} \rightarrow 2\_\{1 - full, 2 - empty\}, 0\_\{1 - full, 2 - empty\} \rightarrow 1\_\{1 - full, 2 - empty\} \rightarrow 2\_\{1 - empty, 2 - empty\}, 0\_\{1 - full, 2 - empty\} \rightarrow 1\_\{1 - full, 2 - empty\}, 0\_\{1 - empty2 - empty\}]$

$test = [0\_\{1 - full, 2 - empty\} \rightarrow 1\_\{1 - full, 2 - empty\}]$

### 3.6.5 Operazioni contenute nel metodo predict

#### Preparazione delle sequenze di test

Per poter partire ad effettuare la predizione, bisogna svolgere le seguenti operazioni di preparazione delle sequenze di test:

| St.Id_test | State | Sequence | N.row |
|---|---|---|---|
| 1 | full | 0_{1-full,2-empty}->1_{1-full,2-empty} | 0 |
| 2 | empty | 0_{1-full,2-empty}->1_{1-full,2-empty} | 0 |

*Tabella 3.2: Contenuto all'interno del oggetto test_df.*

- Filtraggio delle sequenze temporali aventi lunghezza 1, mediante l'utilizzo del metodo filter. Le sequenze temporali aventi lunghezza 1, cioè sequenze costituite solamente da una finestra temporale non sono utili nel test, poiché non hanno altre finestre precedenti per svolgere un match con le sequenze presente nel conseguente delle regole;

  **Esempio 3.6.1.** Nel caso del nostro test set non vi è nessuna sequenza temporale avente lunghezza pari a 1.

- Generazione di un indice, per ogni record, che vada a definire in modo univoco il dato di test. Questa operazione viene effettuata mediante l'utilizzo del metodo zipWithIndex, nell'esempio 3.6.2 si può vedere l'output di questo metodo;

  **Esempio 3.6.2.** Generazione dell'indice presente all'interno del campo valore del Paired RDD, si ottiene come RDD di output il seguente risultato: $(0\_\{1 - full, 2 - empty\} \rightarrow 1\_\{1 - full, 2 - empty\}, 1)$

- Per ogni stazione presente nell'ultima finestra temporale viene generato un oggetto Row, avente la seguente struttura:

  $$St.Id\_test, State, Sequence, N.row$$

  - St.Id_test: definisce il valore di identificativo della stazione di test interessata;
  - State: è lo stato in cui si trova la stazione;
  - Sequence: è la sequenza temporale presa in esame;
  - N.row: è l'identificativo/numero di riga della sequenza.

  Visto che nell'ultima finestra temporale vi può essere più di una stazione, bisogna generare una lista di oggetti Row associati a quella sequenza. Per fare questo si può sfruttare il metodo flatMap, che richiama una funzione transformInRow, che prende come argomento la sequenza di test.

- Trasformazione in DataFrame attraverso l'uso del metodo toDF. Nella tabella 3.2 è possibile vedere la struttura all'interno di questo oggetto.

In seguito si riporta il DAG (figura 3.8) associato a queste trasformazioni per ottenere le manipolazioni relative al test set. Per poter semplificare la spiegazione il DataFrame risultante verrà chiamato test_df.

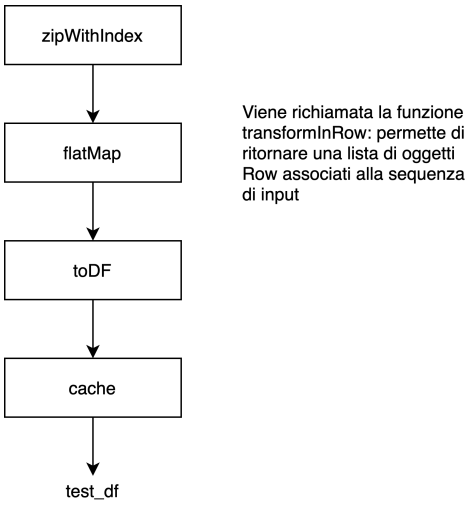


*Figura 3.8: DAG relativo alla preparazione delle sequenze di test.*

### Preparazione del training

Come avviene per le sequenze di test che vengono manipolate per avere una rappresentazione in DataFrame, anche le regole di associazione vengono manipolate; ma esse sono già in formato di DataFrame, dobbiamo cambiare la loro struttura. Per fare questo possiamo delineare i seguenti passaggi:

- Richiamare sul DataFrame contenente le regole di associazione l'attributo rdd che restituisce l'oggetto RDD;
- Applicare la flatMap con la funzione stationExpansion, che prende in input la regola di associazione e crea per ogni stazione presente nel conseguente un oggetto Row avente la seguente struttura:

$$St.Id\_train, antecedent, consequent, St.Pred, conf, lift$$

  - St.Id_train: definisce il valore di identificativo della stazione di training;
  - St.Pred: è lo stato di predizione della stazione;
  - antecedent: è l'antecedente della regola;
  - consequent: è il conseguente della regola;
  - conf: è la confidenza della regola a cui la stazione interessata fa parte;
  - lift: è il lift della regola a cui la stazione interessata fa parte.

- Una volta ottenuti gli oggetti row associati ad ogni stazione presente nel conseguente della regola, bisogna trasformare l'oggetto RDD in DataFrame mediante l'utilizzo del metodo toDF (tabella 3.3).

In seguito si riporta il DAG (figura 3.9) associato a queste trasformazioni per ottenere le manipolazioni relative alle regole di associazione. Per poter semplificare la spiegazione il DataFrame risultante verrà chiamato train_df.

| St.Id_train | antecedent | consequent | St.Pr. | conf | lift |
|---|---|---|---|---|---|
| 1 | 0_{1-full,2-empty} | [2_{1-empty,2-emp... | empty | 0.5 | 2.0 |
| 2 | 0_{1-full,2-empty} | [2_{1-empty,2-emp... | empty | 0.5 | 2.0 |
| 1 | 0_{1-full,2-empty} | [1_{1-full,2-emp... | full | 1.0 | 1.3 |
| 2 | 0_{1-full,2-empty} | [1_{1-full,2-emp... | empty | 1.0 | 1.3 |
| 1 | 0_{1-full,3-full}... | [2_{1-full,2-empty}] | full | 1.0 | 4.0 |
| 2 | 0_{1-full,3-full}... | [2_{1-full,2-empty}] | empty | 1.0 | 4.0 |
| 1 | 0_{1-full,2-empty... | [2_{1-empty,2-emp... | empty | 0.5 | 2.0 |
| 2 | 0_{1-full,2-empty... | [2_{1-empty,2-emp... | empty | 0.5 | 2.0 |
| 1 | 0_{1-full,3-full} | [1_{1-full,2-emp... | full | 1.0 | 1.3 |
| 2 | 0_{1-full,3-full} | [1_{1-full,2-emp... | empty | 1.0 | 1.3 |
| 1 | 0_{1-full,3-full} | [2_{1-full,2-empty}] | full | 1.0 | 4.0 |
| 2 | 0_{1-full,3-full} | [2_{1-full,2-empty}] | empty | 1.0 | 4.0 |

*Tabella 3.3: Contenuto all'interno del oggetto train_df.*

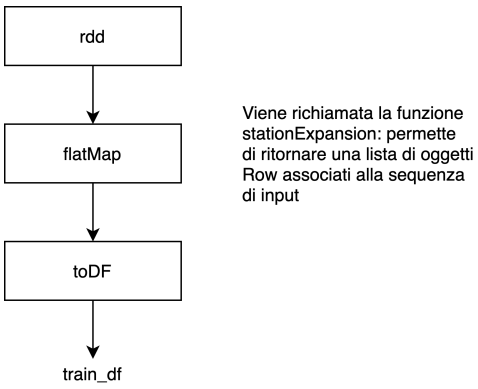


*Figura 3.9: DAG relativo alla preparazione delle sequenze di training.*

### Join e filtro sul conseguente

Per mettere in relazione le stazioni che sono contenute nei DataFrame, risultato dell'operazione di preparazione, chiamati test_df e train_df, possiamo utilizzare il metodo join.

Questo metodo compie il join tra due DataFrame, usando, in questo caso, l'identificativo della stazione come vincolo. Il risultato finale consiste nel seguente formato di uscita:

$$St.Id\_test, State, Sequence, N.row, St.Id\_train, antecedent, consequent,$$

$$St.Pred., conf, lift$$

Tuttavia, in questo risultato (tabella 3.4) non vi sono stazioni di test che non sono presenti nel tain_df, e viceversa per le stazioni di training che non sono presenti nel test. Nel primo caso, quelle stazioni che non sono presenti nel train_df dovranno essere classificate con lo stato Normal (questo verrà affrontato nei prossimi punti).

Il passo successivo consiste nell'imporre la condizione del vincolo sull'antecedente, ciò viene fatto con l'utilizzo di una UDF di nome udfCheckAntecedent, che ritorna un booleano: se la condizione non è verificata assume valore false, altrimenti true.

Questa UDF viene richiamata all'interno del metodo where che si occupa di filtrare i record che non soddisfano il vincolo (il risultato di questa operazione è presente nella tabella 3.5).

| St.Id_test | State | Sequence | N.Row | St.Id_train | antecedent | consequent | St.Pred | conf | lift |
|---|---|---|---|---|---|---|---|---|---|
| 1 | full | 0_{1-full,2-empty... | 0 | 1 | 0_{1-full,2-empty} | [1_{1-full,2-emp... | full | 1.0 | 1.3 |
| 1 | full | 0_{1-full,2-empty... | 0 | 1 | 0_{1-full,2-empty} | [2_{1-empty,2-emp... | empty | 0.5 | 2.0 |
| 2 | empty | 0_{1-full,2-empty... | 0 | 2 | 0_{1-full,2-empty} | [2_{1-empty,2-emp... | empty | 0.5 | 2.0 |
| 1 | full | 0_{1-full,2-empty... | 0 | 1 | 0_{1-full,3-full}... | [2_{1-full,2-empty}] | full | 1.0 | 4.0 |
| 2 | empty | 0_{1-full,2-empty... | 0 | 2 | 0_{1-full,3-full}... | [2_{1-full,2-empty}] | empty | 1.0 | 4.0 |
| 1 | full | 0_{1-full,2-empty... | 0 | 1 | 0_{1-full,2-empty... | [2_{1-empty,2-emp... | empty | 0.5 | 2.0 |
| 2 | empty | 0_{1-full,2-empty... | 0 | 2 | 0_{1-full,2-empty... | [2_{1-empty,2-emp... | empty | 0.5 | 2.0 |
| 1 | full | 0_{1-full,2-empty... | 0 | 1 | 0_{1-full,3-full} | [2_{1-full,2-empty}] | full | 1.0 | 4.0 |
| 2 | empty | 0_{1-full,2-empty... | 0 | 2 | 0_{1-full,3-full} | [2_{1-full,2-empty}] | empty | 1.0 | 4.0 |
| 1 | full | 0_{1-full,2-empty... | 0 | 1 | 0_{1-full,3-full} | [1_{1-full,2-emp... | full | 1.0 | 1.3 |

*Tabella 3.4: Contenuto del dataftame risultante dopo l'operazione di join tra train_df e test_df.*

| St.Id_test | State | Sequence | N.Row | St.Id_train | antecedent | consequent | St.Pred | conf | lift |
|---|---|---|---|---|---|---|---|---|---|
| 1 | full | 0_{1-full,2-empty... | 0 | 1 | 0_{1-full,2-empty} | [2_{1-empty,2-emp... | empty | 0.5 | 2.0 |
| 2 | empty | 0_{1-full,2-empty... | 0 | 2 | 0_{1-full,2-empty} | [2_{1-empty,2-emp... | empty | 0.5 | 2.0 |

*Tabella 3.5: Contenuto del dataftame risultante dopo l'operazione di filtraggio.*

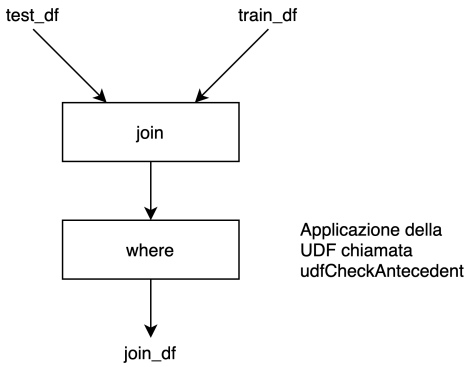


*Figura 3.10: DAG relativo all'operazione di join e di filtraggio sull'antecedente.*

Nella figura 3.10, è possibile vedere il DAG relativo a queste operzioni descritte, il DataFrame risultante viene denominanto semplicemente come join_df.

### Creazione dello score

Nel caso in cui si voglia predire lo stato utilizzando più regole di associazione in modo pesato, bisogna introdurre una metodologia per poter capire quali siano le regole più rilevanti per la predizione di quel dato di test. Ciò può essere fatto mediante l'utilizzo di uno score. Nella creazione dello score relativo ad una regola di associazione, bisogna effettuare come prima cosa la normalizzazione dei campi di confidenza e di lift, poiché queste due misure hanno un range di valori diversi e non possono essere valutati in modo semplice. Il tipo di normalizzazione scelta è quella definita dal MinMaxScalar:

$$X_{normalizzato} = \frac{X_i - min(X)}{max(X) - min(X)}, \forall i \in K$$

La normalizzazione viene svolta utilizzando la libreria MLib di Spark, che permette di definire una pipeline con le operazioni di Machine Learning che si vogliono effettuare sui dati.

Una volta ottenuti i dati normalizzati possiamo creare una nuova colonna in cui definire la media tra le nuove colonne di confidenza e di lift normalizzate, come si può vedere nella tabella 3.6.

| **St.Id_test** | **State** | **Sequence** | **N.Row** | **St.Id_train** | **St.Pred** | **conf** | **lift** | **conf_Scaled** | **lift_scaled** | **score** |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | full | 0_{1-full,2-empty... | 0 | 1 | empty | 0.5 | 2.0 | 0.5 | 0.5 | 1 |
| 2 | empty | 0_{1-full,2-empty... | 0 | 2 | empty | 0.5 | 2.0 | 0.5 | 0.5 | 1 |

*Tabella 3.6: Contenuto del DataFrame risultante dopo l'operazione di normalizzazione.*

| St.Id_test | Sequence | N.row | State | k_list | St.Pred. |
|---|---|---|---|---|---|
| 1 | 0_{1-full;2-empty... | 0 | full | [[empty;1.0]] | empty |
| 2 | 0_{1-full;2-empty... | 0 | empty | [[empty;1.0]] | empty |

*Tabella 3.7: Contenuto del DataFrame risultante dopo l'operazione di predizione.*

### Ordinamento per importanza

Come spiegato in precedenza, bisogna introdurre all'interno nel nostro DataFrame risultante un ordinamento delle righe associate all'identificativo della stazione di cui si vuole predire lo stato, stazione che si trova all'interno di una sequenza temporale (definita dall'indice di riga creato in precedenza). Per fare questo si può creare una partizione sui campi di StationId_test, sequence, num_row. All'interno di ogni partizione si compie l'ordinamento sui campi di:

- Confidenza;
- Lift;
- Lunghezza della regola;
- Ordine lessico grafico derivato dall'antecedente.

Il prossimo passo da effettuare è quello di selezionare solo K righe all'interno della partizione. La partizione in questione è quella in cui viene utilizzata la funzione row_number, che enumera tutte le righe all'interno. Successivamente viene posto il vincolo sul numero di riga, che deve essere minore o uguale al valore di K.

In seguito, i dati all'interno della partizione vengono aggregati andando a formare una lista avente al suo interno lo stato e lo score associato al record. Per poter mettere insieme questi due valori si utilizza la funzione struct che crea un vettore di due elementi:

- Il primo elemento definisce lo stato;
- Il secondo elemento definire lo score associato.

### Predizione

La predizione effettiva viene fatta attraverso la creazione di una nuova colonna di nome StatePredicition, che viene creata attraverso l'utilizzo del metodo withColumn che richiama la funzione predictUdf. Quest'ultima permette di compiere la predizione utilizzando i parametri di configurazione descritti all'interno dell'oggetto ConfigurationClassifier e il campo score del DataFrame in modo opportuno in base alla tipologia di classificazione. Questa funzione ritorna lo stato predetto per la relativa riga all'interno del DataFrame (tabella 3.7).

Per poter gestire le stazioni che non sono presenti all'interno del conseguente delle regole di associazioni, possiamo fare, in ordine:

- Left-anti join: che permette di selezionare i dati appartenenti al test_df che non sono contenuti all'interno del DataFrame risultante. I vincoli per questo join saranno StationId_test e la colonna di nome Num_row. Nel caso del nostro esempio considerato il DataFrame in output sarà vuoto;

| StationId_test | Sequence | Num_row | State | StatePred. |
|---|---|---|---|---|
| 1 | 0_{1-full;2-empty... | 0 | full | empty |
| 2 | 0_{1-full;2-empty... | 0 | empty | empty |

*Tabella 3.8: Contenuto del DataFrame finale dopo l'operazione di predizione.*

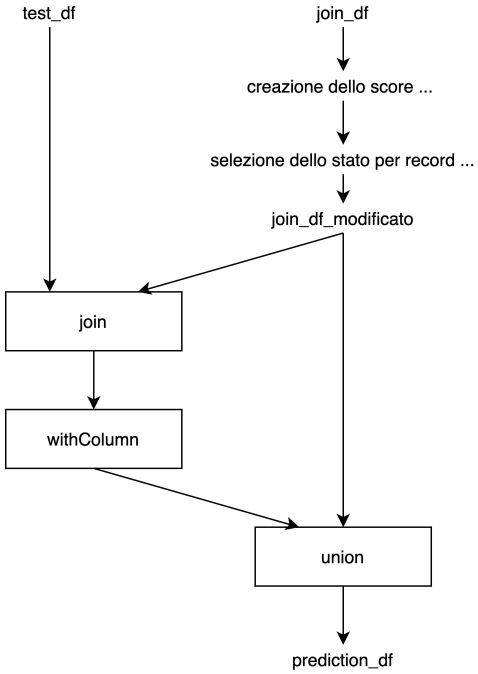


*Figura 3.11: DAG relativo alle operazioni finali per ottenere la predizione.*

- Inserire nel DataFrame ottenuto la colonna di nome StatePrediciton con il valore pari a Normal, anche in questo caso dell'esempio il DataFrame risultante sarà vuoto;
- Per poter unire il DataFrame risultante con le altre predizioni basta effettuare una union, che permette di unire le due righe in un unico DataFrame. Il DataFrame risultante verrà chiamato prediciton_df. Nel nostro esempio si otterrà il seguente DataFrame rappresentato dalla tabella 3.8.

Nella figura 3.11 è possibile vedere le operazioni finali compiute per la computazione delle predizione sugli stati, in output si ottiene il DataFrame denominato prediction_df.

### Statistiche sulle predizioni

Una volta compiuta la predizione dei dati di test, si possono calcolare le statistiche relative.

Come prima cosa bisogna trasformare il nostro DataFrame in un Paried RDD, avente la seguente struttura:

$$(StationID, prediction\ real\_state).$$

Il passo successivo sarà di raggruppare tutti i risultati relativi alle stazioni facendo una reduceByKey, inserendo nel campo valore una stringa composita di tutte

le predizioni, legate dal carattere speciale "-". A questo consegue il calcolo delle misure di interesse per poter valutare le performance della predizione. Le misure calcolate sono:

- Accuratezza;
- Precisione per gli stati Full, Empty e Normal;
- Richiamo per gli stati Full, Empty e Normal;
- $F_1$ score per gli stati Full, Empty e Normal;
- Matrice di confusione.

Questi risultati vengono strutturati in una stringa così formata:

$$
\begin{aligned}
&StationId,\\
&precisione\_full, precisione\_empty, precisione\_normal,\\
&richiamo\_full, richiamo\_empty, richiamo\_normal,\\
&F1\_score\_full, F1\_score\_empty, F1\_score\_normal,\\
&matrice\ di\ confusione.
\end{aligned}
$$

Un altro punto che bisogna affrontare, il calcolo delle statistiche generali di tutte le misure calcolate per ogni stazione, per poi accorparle calcolando per ognuna di essa i seguenti valori:

- Valore medio;
- Minimo;
- Massimo.

Per il calcolo del valore medio, si sommano tutte le misure compiendo l'operazione di reduce utilizzando la funzione sumStatistic, e in parallelo si possono calcolare anche il valore di minimo e massimo utilizzando la funzione computeMinMax sempre utilizzando il metodo reduce. Una volta ottenuta la somma, i minimi e i massimi delle misure, possiamo unire questi due valori facendo una parallelize, introducendoli all'interno di un vettore e, in seguito, utilizzando il metodo map si può calcolare effettivamente la media solo per il valore di somma. I risultati ottenuti verranno salvati all'interno del HDFS.

Nella figura 3.12 è presente il DAG relativo al calcolo delle statisctiche di base.

## 3.7 Classificatore basato su timeslot

### 3.7.1 Adattamento delle sequenze

Per introdurre la classificazione basata sul timeslot bisogna riadattare le nostre sequenze introducendo l'informazione del timestamp di partenza.

L'adattamento delle sequenze temporali comporta una modifica del metodo generate_sequence presente all'interno della classe GenerateDataset, rimuovendo l'operazione di values, che permette di estrarre il valore presente all'interno dell'Paired

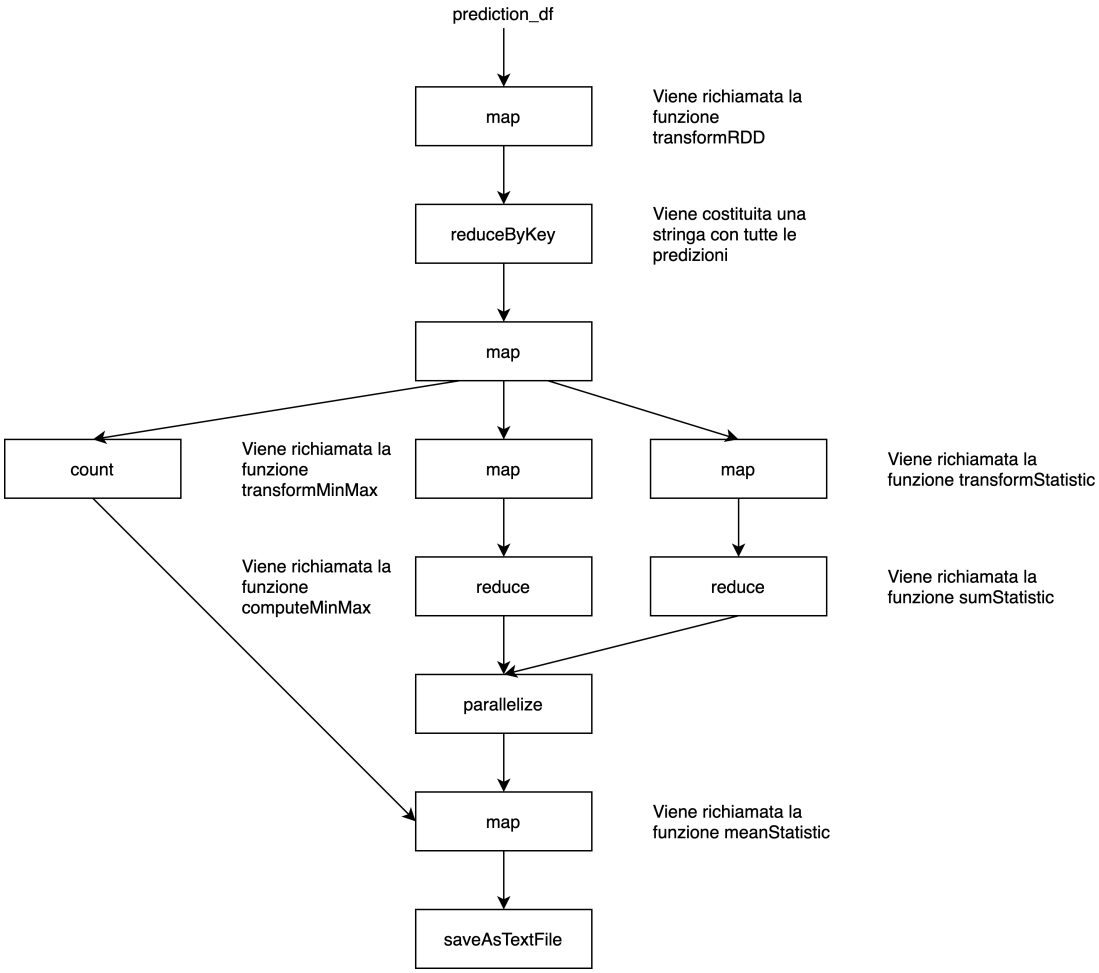


*Figura 3.12: DAG corrispondente al calcolo delle statistiche di base.*

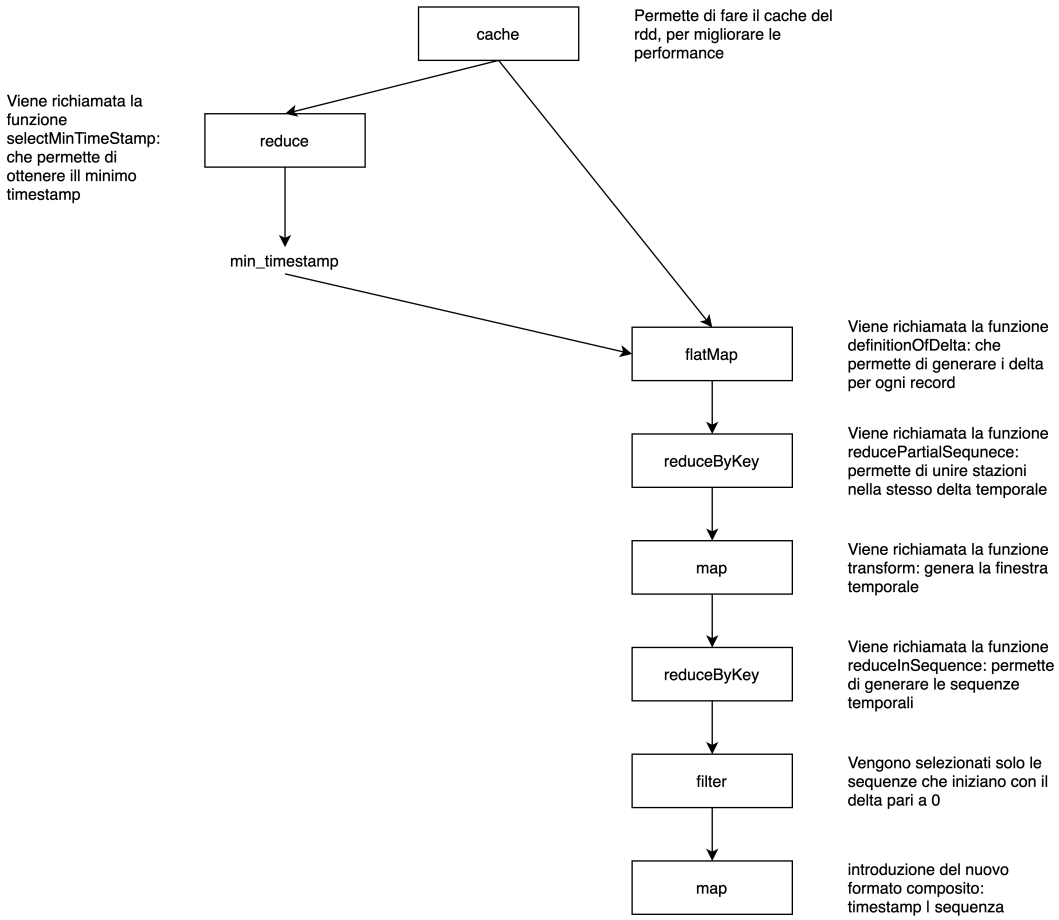


*Figura 3.13: DAG relativo alle modifiche apportate al metodo generate_sequence.*

RDD. Aggiungendo subito dopo il filtraggio, l'operazione di map crea una stringa composta nel seguente modo:

$$timestamp|sequenza.$$

Nella figura 3.13 è possibile il DAG del metodo generate_sequence aggiornato.

### 3.7.2 Implementazione del classificatore

Per classificare i nostri dati di test specifici per il timeslot, bisogna estrarre diverse regole di associazione relative a diverse fasce orarie e andare a selezionare il corrispondente modello in accordo con il timestamp di partenza.

Questa tipologia di classificazione viene fatta dalla classe TimeSlotClassifier, che al suo interno riceve i parametri su cui andrà a creare le regole di associazione e i parametri relativi alla costruzione di diversi oggetti Classifier, corrispondenti alle fasce orarie.

Nella figura 3.14 è possibile vedere il risultato finale di tutte le relazioni che vi sono tra le classi e anche l'introduzione della classe TimeSlotClassifier.

Come si può notare, il nostro nuovo classificatore è associato ad un parametro definito come un vettore di tuple corrispondenti all'ora di inizio e all'ora di fine della fascia oraria. Questo parametro può essere soggetto ad un'analisi per comprendere qual è il valore più appropriato da scegliere per poter effettuare la classificazione.

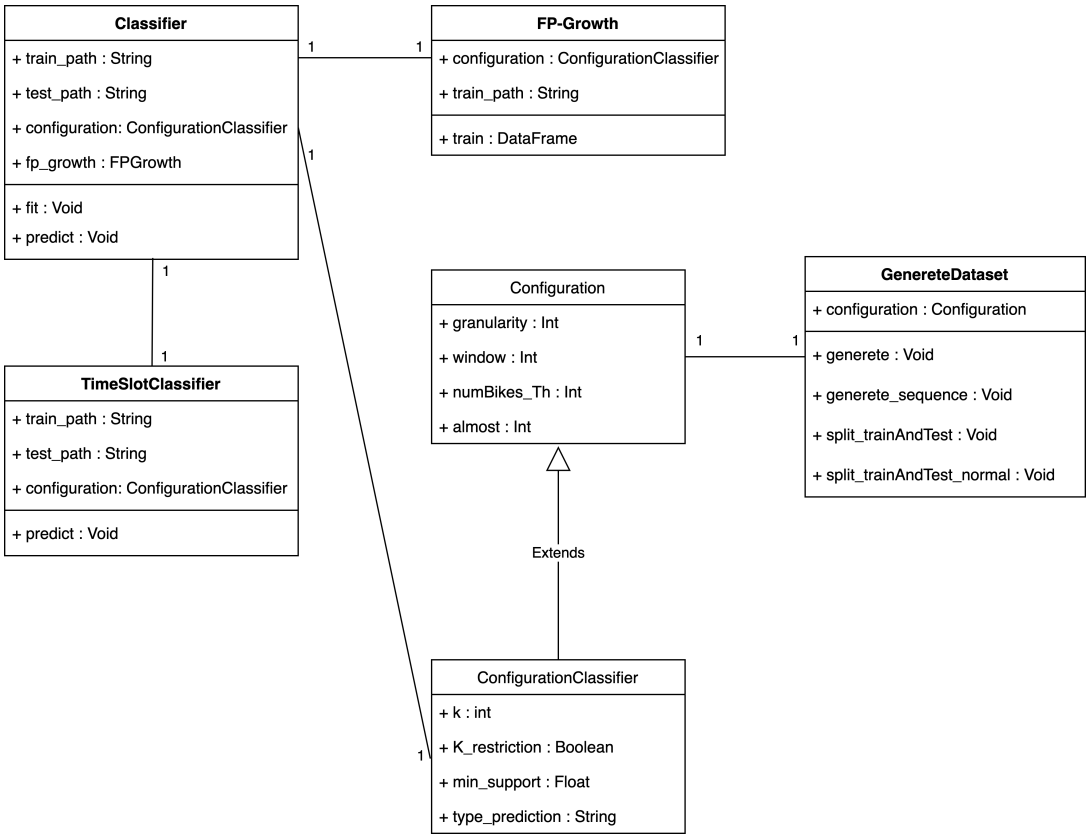


*Figura 3.14: UML: GenerateDataset, Configuration, Classifier, TimeslotClassifier e ConfigurationClassifier.*

Di default le fasce orarie sono quattro:

- dalle ore 4:00 alle ore 9:59;
- dalle ore 10:00 alle ore 15:59;
- dalle ore 16:00 alle ore 21:59;
- dalle ore 22:00 alle ore 3:59.

Nel metodo predict, si effettua un ciclo for sul vettore che costituisce i valori delle fascie orarie e si selezionano i dati di training e di test che appartengono a quella fascia oraria. La selezione viene fatta attraverso il metodo filter che richiama al suo interno una funzione di nome checkTimeslot, la quale ritorna un booleano a seconda se il dato rientra nella fascia oraria.

Il training e il test set relativi ad una determinata fascia oraria vengono salvati all'interno del HDFS e successivamente i path andranno in input all'oggetto Classifier. Le predizioni ottenute, per ogni fascia oraria, possono essere raggruppate utilizzando il metodo union poi si può compiere il calcolo delle statistiche come descritte in precedenza.

# Capitolo 4

# Valutazione del lavoro

## 4.1 Esplorazione del dataset

L'esplorazione del dataset è la prima e la più importante analisi da effettuare. Si parte dal rilevare qual è il numero di record presenti nel dataset, che è 25319028.

### 4.1.1 Analisi sulle stazioni

Il numero di stazioni presenti nel dataset è 284, il numero medio (e la varianza) di record per stazione è pari a:

$$89151.5070 \pm 5905.8730.$$

Il numero massimo di record associati ad una singola stazione è 89796, le stazioni aventi questo numero di record sono:

$$280,\ 281,\ 282,\ 283.$$

Il numero minimo di record presenti per una singola stazione è 166. La stazione associata al minor numero di record è la stazione numero 284.

### 4.1.2 Analisi sulla grandezza delle stazioni

Si può effettuare uno studio riguardante la grandezza massima che ogni stazione può assumere, così facendo possiamo generare il seguente grafico (figura 4.1) che mostra come varia la frequenza in base alla grandezza delle stazioni. Questo grafico può anche essere rappresentato come distribuzione (figura 4.1).

### 4.1.3 Analisi sui valori di used e free

Il prossimo studio è quello relativo ai campi di used e di free presenti nel nostro dataset. Nella figura 4.2, possiamo visionare la frequenza dei valori di used e la sua distribuzione comulativa per capire come è distribuita la frequenza dei valori che il campo used può assumere.

Nella figura 4.3, possiamo vedere la frequenza dei valori assunti dal campo free e la loro relativa distribuzione comulativa.

Dalla rappresentazione grafica, presente nelle figure 4.2 e 4.3, si evince un picco sullo 0. Nel grafico relativo ai valori di used, si può notare un andamento decrescente

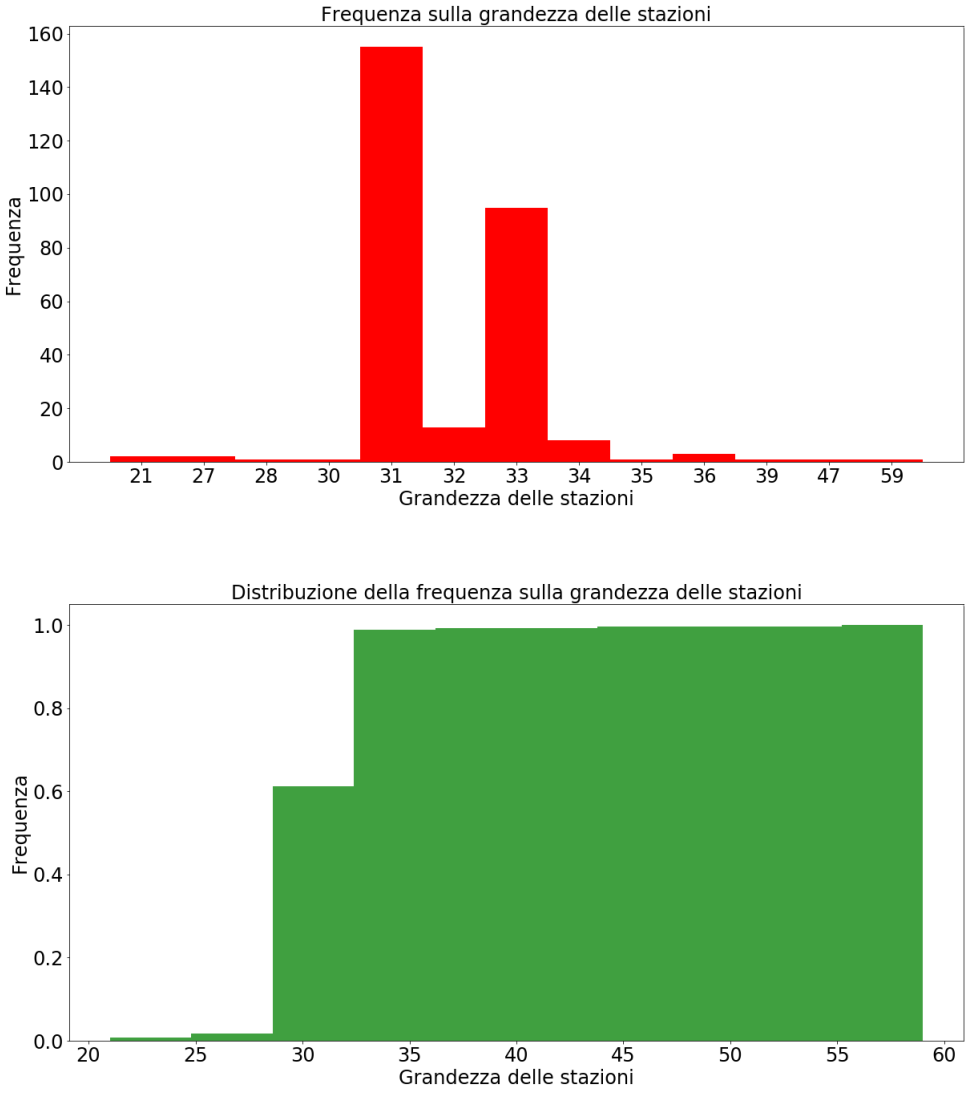


*Figura 4.1: Frequenza sulla grandezza delle stazioni.*

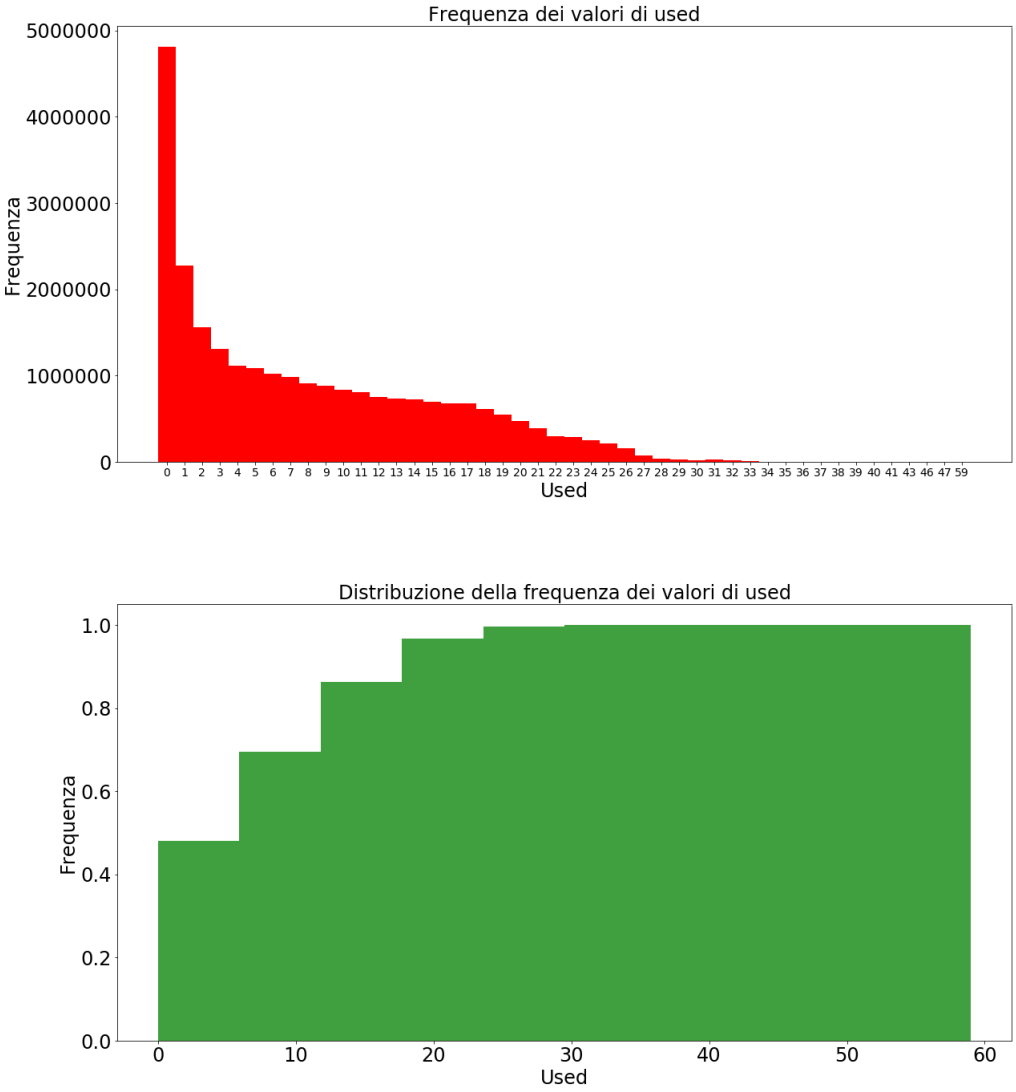


*Figura 4.2: Frequenza dei valori di used.*

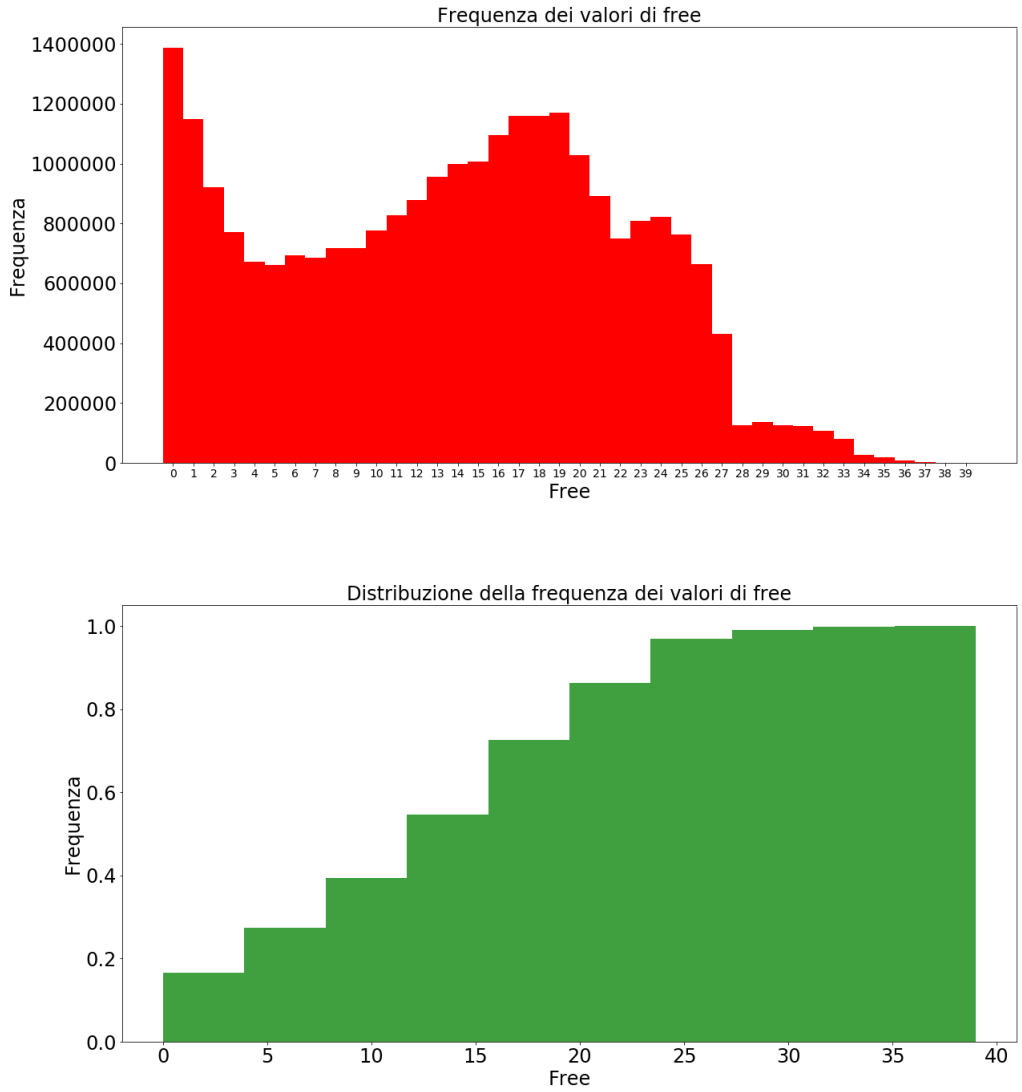


*Figura 4.3: Frequenza dei valori di free.*

con l'aumentare del valore di used. L'andamento decrescente ci può far pensare che le stazioni abbiano una situazione di stato Empty in molti casi, contrariamente gli ordini di grandezza dell'asse delle y è pari a 1000000. Tutte le altre situazioni in cui il campo used è diverso da 0, ne deriva lo stato predominante nel nostro dataset è Normal.

Considerando il campo free si nota un andamento decrescente fino al valore di 8 e un aumento fino ad arrivare ad un massimo locale per il valore di 18 ed una successiva decrescita. Anche in questo caso il grafico ci può portare a pensare che la situazione più frequente è quella in cui lo stato è Full, mentre qui è più evidente che lo stato predominante è quello di Normal.

### 4.1.4 Analisi sugli stati

Un altro caso di interesse consiste nell'analizzare la frequenza dei record che ricadono nello stato di Full, Empty e Normal. Tale informazione è riportata nel grafico

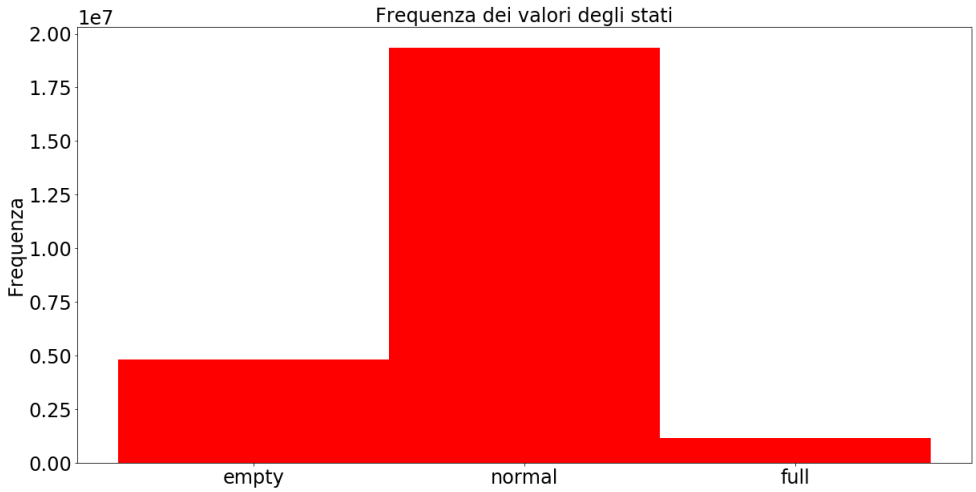


*Figura 4.4: Frequenza delle stazioni negli stati: Full, Empty e Normal.*

rappresentato dalla figura 4.4.

Inoltre, possiamo estendere l'analisi anche al caso di AlmostFull e AlmostEmpty. Possiamo impostare l'analisi con valori di soglia (chiamata numBikes_Th da qui in avanti) pari a 2 (figura 4.5), 4 (figura 4.6) e 6 (figura 4.7), per sperimentare come varia la frequenza degli stati di Normal, AlmostFull e AlmostEmpty.

Come si può evincere dai grafici descritti in precedenza, vi è un picco di situazioni in cui le stazioni ricadono nel caso di Normal, tranne in numBikes_Th pari a 6, per cui si ottiene un picco sullo stato AlmostEmpty.

In tutti questi casi, il picco che si genera sullo stato Normal e sullo stato AlmostEmpty, ci permette di dedurre che il nostro dataset genera degli stati sbilanciati e può generare un problema durante la classificazione, poiché vi sono meno casi di stati Empty e Full da cui apprendere.

### 4.1.5 Analisi sul timestamp

Visto che la nostra analisi si basa sul concetto di tempo, è utile vedere come si comporta il valore di timestamp nel nostro dataset. Il numero di timestamp distinti è pari a 89802, con valore minimo:

$$2008 - 05 - 15\ 12 : 01 : 00.$$

Il valore massimo di timestemp è pari a:

$$2008 - 09 - 30\ 23 : 58 : 00.$$

Il valore medio di record che ricadono in un timestamp generico è pari a:

$$281.9428 \pm 4.1161.$$

Il numero massimo di record che un timestamp riesce ad ottenere è a 283 (che è quasi uguale al valore del numero di stazioni), il numero di timestamp che assumono

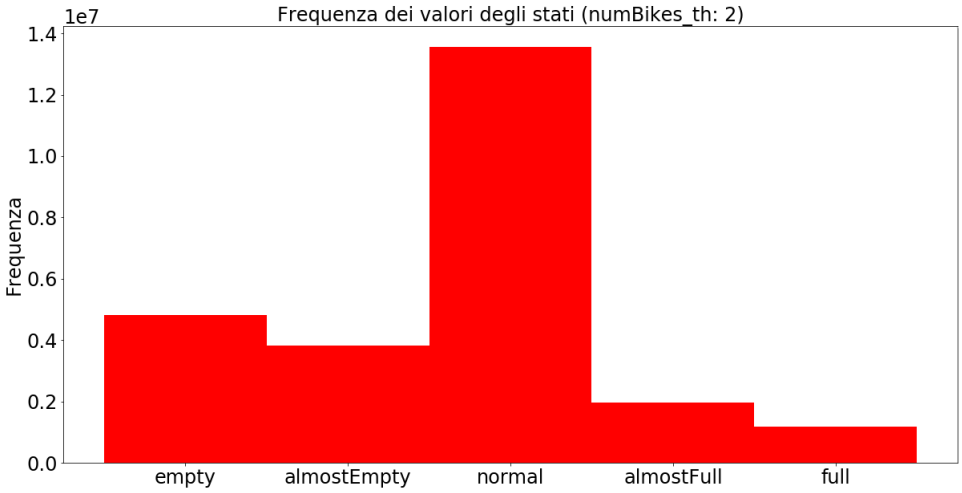


*Figura 4.5: Frequenza delle stazioni negli stati: Full, Empty, Normal, AlmostFull e AlmostEmpty nel caso di numBikes_Th pari a 2.*

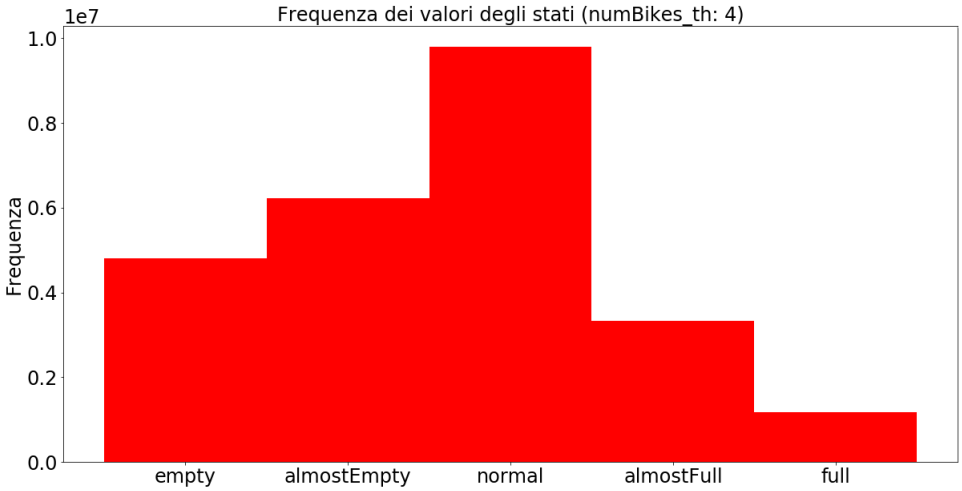


*Figura 4.6: Frequenza delle stazioni negli stati: Full, Empty, Normal, AlmostFull e AlmostEmpty nel caso di numBikes_Th pari a 4.*

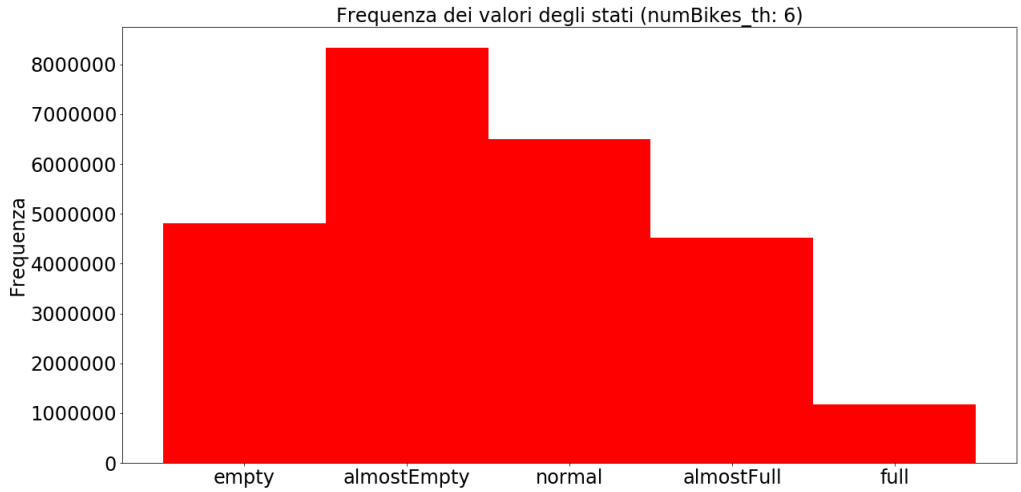


*Figura 4.7: Frequenza delle stazioni negli stati: Full, Empty, Normal, AlmostFull e AlmostEmpty nel caso di numBikes_Th pari a 6.*

questo valore è pari a 53848. Il numero di record minimo che ricade in un timestamp è 1, ed è caratterizzato dal timestamp di valore: 2008-09-04 16:08:00.

Possiamo rappresentare la variazione del numero di biciclette (figura 4.8) con i valori di free e used per ogni istante di timestamp per visualizzare in maggior dettaglio l'andamento di questi valori. Nel caso considerassimo il valore del numero di biciclette per istante di tempo, osserveremmo che al variare del timestamp esso non rimane coerente. Nel caso ideale, la somma dei due valori dovrebbe essere costante al variare del timestamp.

Nel dettaglio possiamo visualizzare anche la variazione in base alle ore della giornata per capire come si comporta il dataset, figura 4.9.

Nella figura 4.10 si visualizza la variazione dei minuti in un giorno in modo progressivo.

## 4.2 Analisi delle regole di associazione

Un altro comportamento utile da comprendere è quello relativo alla variazione del minimo supporto per l'estrazione delle regole.

E' possibile effettuare questa analisi sia su tutte le sequenze temporali sia quelle relative al training set di interesse.

Nei grafici riportati si può notare un andamento decrescente in scala logaritmica (con base 10) sull'asse delle ordinate (figure 4.11 e 4.12).

Il valore minimo di supporto più significativo è 0.0001 e possiamo visualizzare i seguenti grafici:

- La distribuzione della frequenza della lunghezza delle sequenze presenti nell'antecedente della regola;
- Valori di lunghezza media, minimo e massima delle sequenze che hanno le prime dieci stazioni presenti nel conseguente della regola;

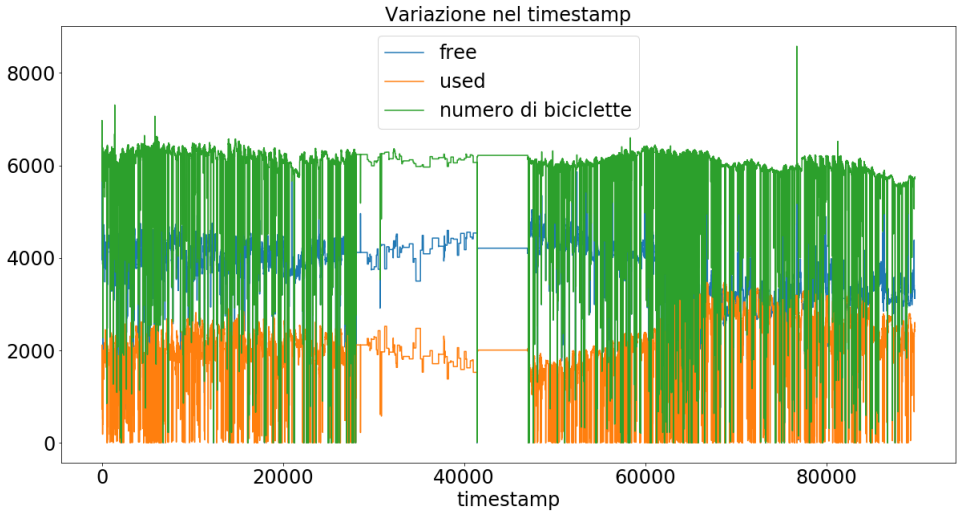


*Figura 4.8: Frequenza dei valori di free, used e numero di biciclette nella variazione dell'intero range di valori del timestamp.*

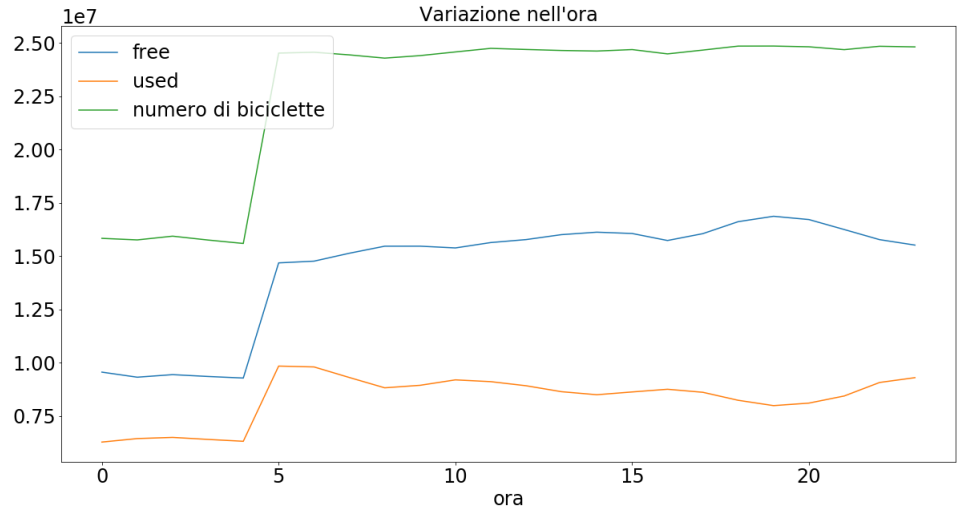


*Figura 4.9: Frequenza dei valori di free, used e numero di biciclette nella variazione dell'intero range di ore che vi sono in un giorno.*

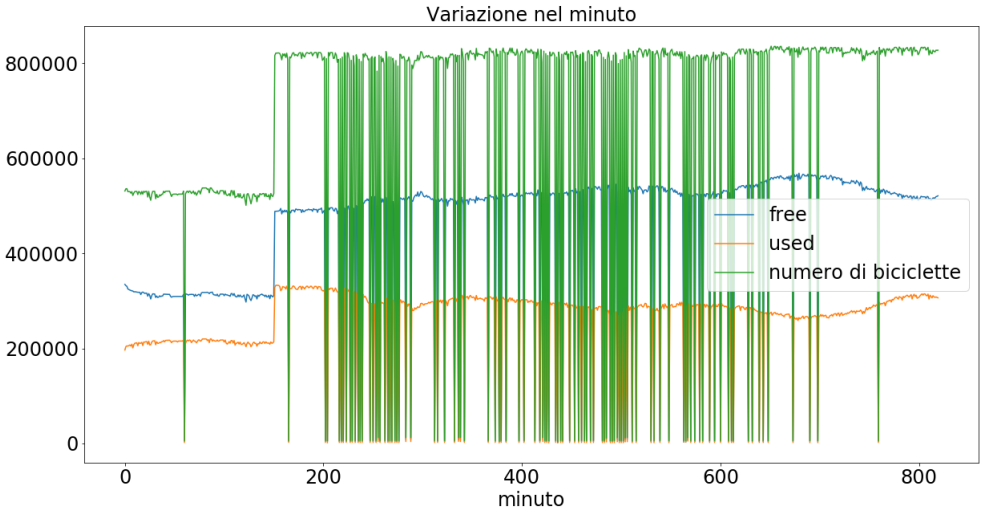


*Figura 4.10: Frequenza dei valori di free, used e numero di biciclette nella variazione dell'intero range di minuti che vi sono in un giorno.*

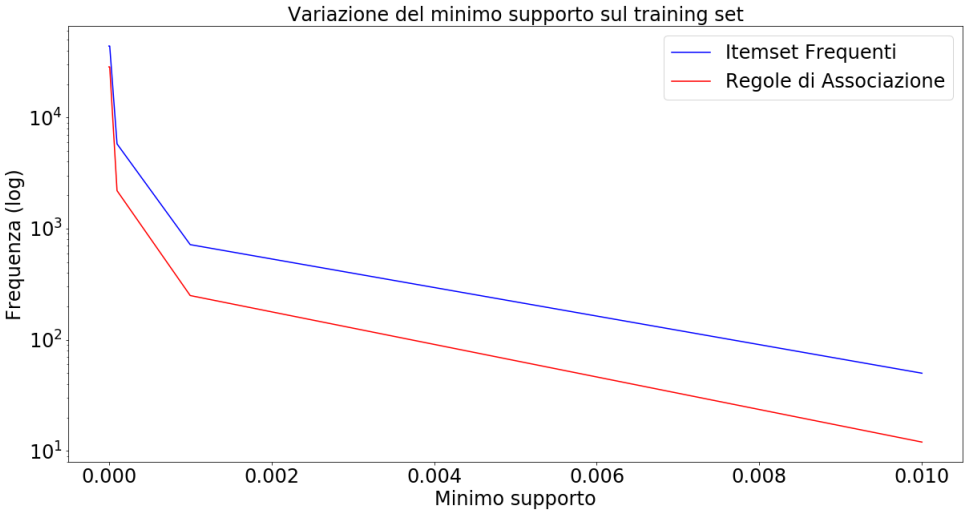


*Figura 4.11: Variazione della soglia di minimo supporto sul training set.*

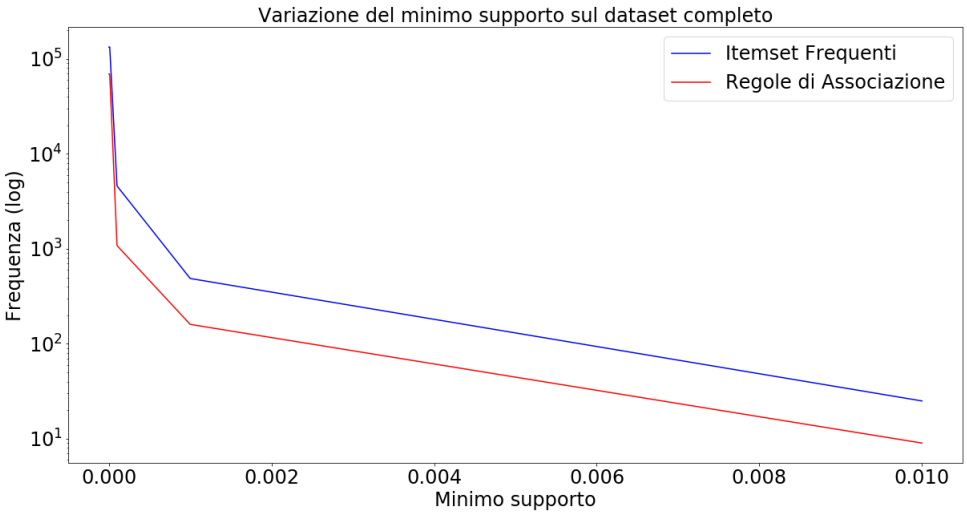


*Figura 4.12: Variazione della soglia di minimo supporto sul dataset intero.*

- Distribuzione dei valori di confidenza minima, massima e media;
- Distribuzione dei valori di lift minimo, massimo e medio.

Come si evince dalla figura 4.13, si ottiene un picco nella distribuzione pari al numero medio di lunghezza pari a 1.30. Questa distribuzione ricorda una gaussiana, poiché attorno al valore di picco più alto vi è una diminuzione del valore medio della lunghezza delle stazioni. Nel secondo grafico sempre relativo alla figura 4.13, si può notare che le prime dieci stazioni presentano come valore medio pari a 1.30, come valore di massimo 2 (poiché stiamo considerando sequenze di lunghezza uguale a 3, quindi nell'antecedente della regole si può avere come lunghezza massima 2) e come valore di minimo 1.

Nella figura 4.14 si può osservare la distribuzione dei valori massimo, minimo e medio della confidenza delle regole estratte e nel grafico successivo il valore massimo, minimo e medio relativo alle prime dieci stazioni.

Nella figura 4.15, invece, si vedono le distribuzioni dei valori massimi, minimi e medi del lift che le regole di associazione possono assumere e di conseguenza vengono riportati i valori medi, minimi e massimi relativi alle prime dieci stazioni.

## 4.3 Confronto dei classificatori sulle prime dieci stazioni

In questo punto del capitolo si compiono dei confronti tra i classificatori, sulle loro predizioni, prendendo in esame le prime dieci stazioni.

### 4.3.1 Variazione della granularità

Per capire come si comporta la nostra classe Classifier sul dataset, dobbiamo prima capire quale possa essere il valore ideale da utilizzare come parametro di granularità per la generazione delle sequenze. Quindi dobbiamo generare diversi insiemi di

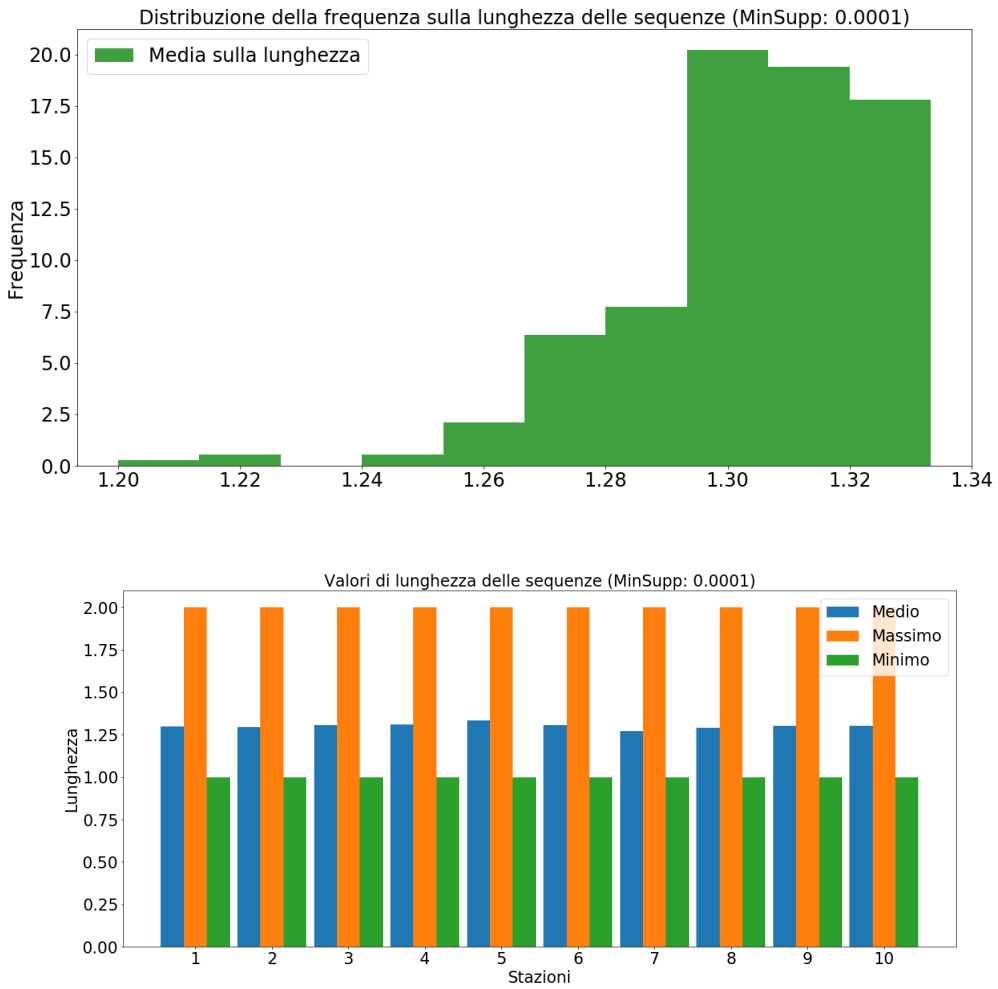


*Figura 4.13: Distribuzione della lunghezza media delle regole di associazione e lunghezza media, minima e massima delle sequenze nelle regole di associazione in cui ricadono le prime dieci stazioni.*

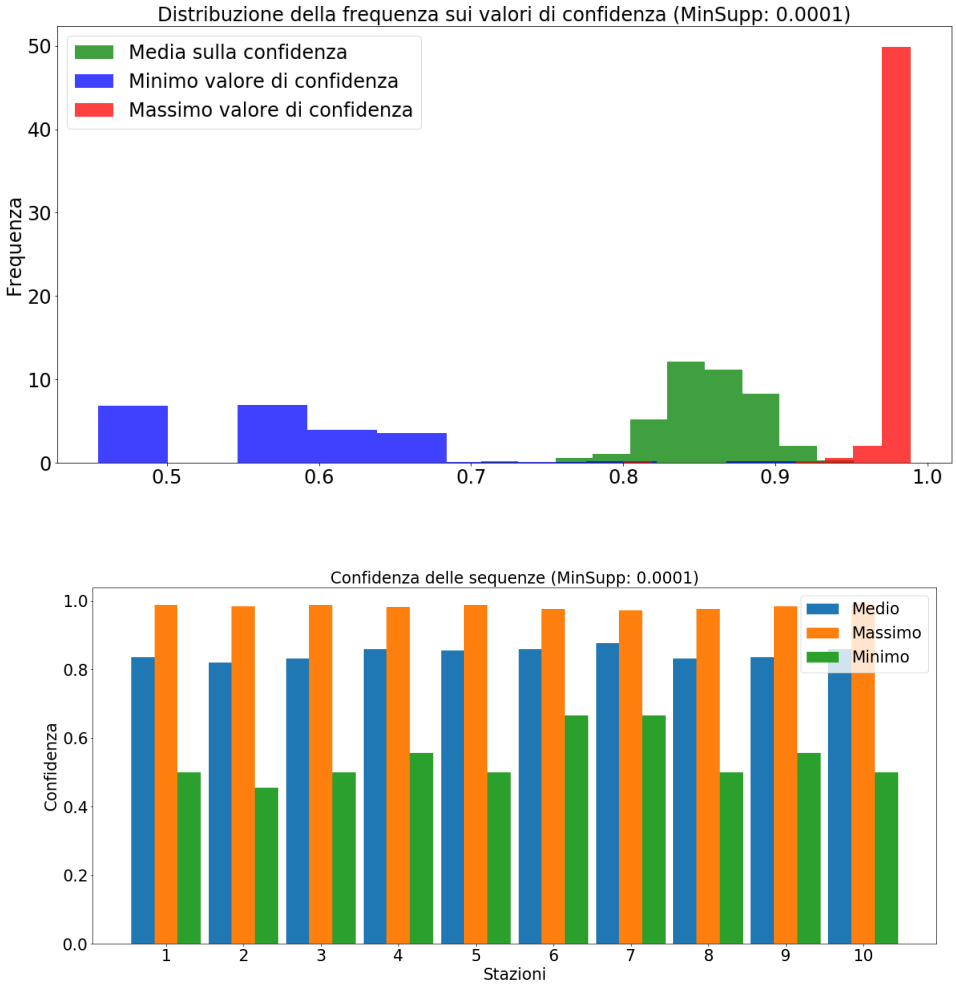


*Figura 4.14: Distribuzione della confidenza delle regole di associazione e confidenza media, minima e massima delle sequenze nelle regole di associazione in cui ricadono le prime dieci stazioni.*

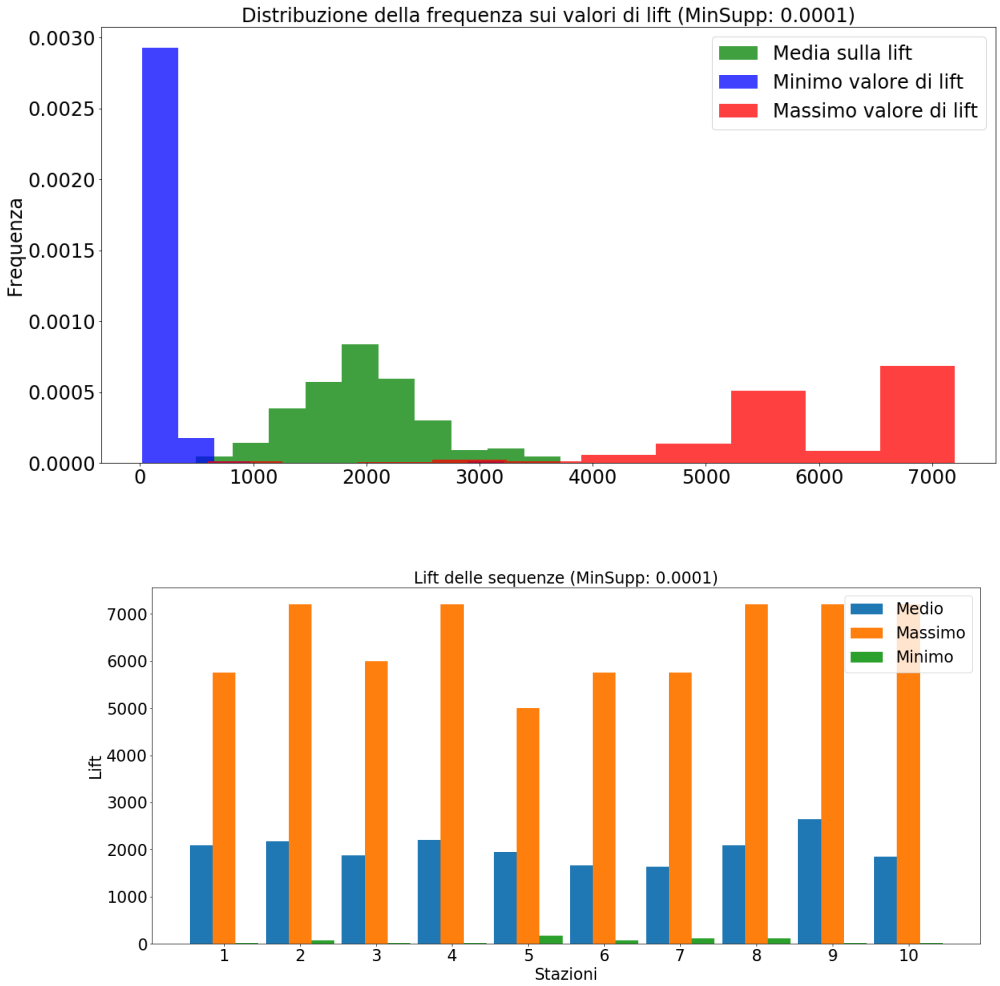


*Figura 4.15: Distribuzione del lift delle regole di associazione e lift media, minima e massima delle sequenze nelle regole di associazione in cui ricadono le prime dieci stazioni.*

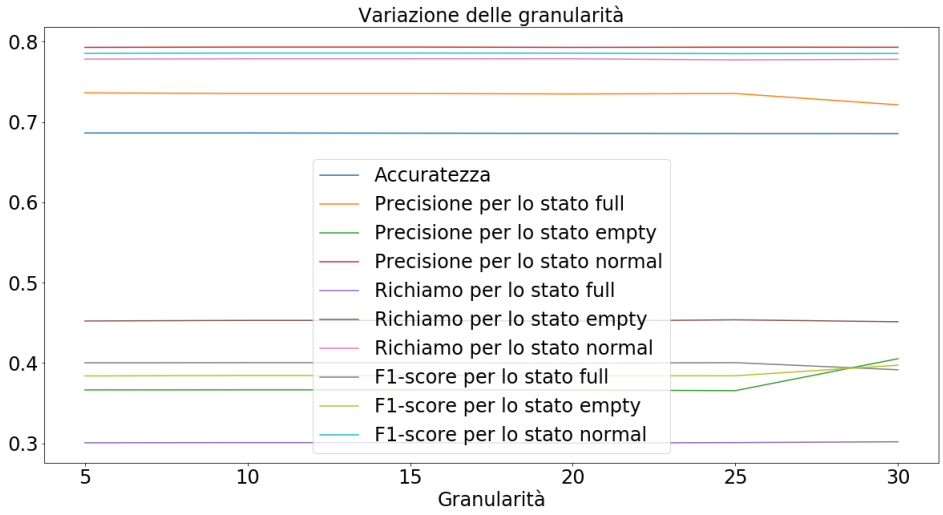


*Figura 4.16: Variazione della granularità nel caso delle prime dieci stazioni.*

dataset creati ognuno con un valore diverso di granularità e verificare come tale scelta impatta sulla qualità delle predizioni.

I valori di granularità che andremo a studiare sono pari a 5, 10, 15, 20, 25 e 30 minuti.

In questa analisi si selezionano i seguenti parametri di classificazione:

- K: 1;
- type_prediction: majority;
- K_restriction: false.

Nella figura 4.16, si nota come l'andamento delle misure decresce leggermente con l'aumentare del valore di granularità, tranne per il caso della misura di precisone dello stato Empty, che con un picco nel valore di granularità pari a 30 minuti. Nella figura 4.17 è possibile vedere come decresce l'accuratezza all'aumentare della granularità.

### 4.3.2 Variazione del valore di numBikes_Th

In questa analisi si prende in considerazione il valore di granularità pari a 5 minuti, poiché i risultati in generale rispetto ad altri valori non cambiano di molto. Tendo fisso il valore di granularità, possiamo variare il valore di numBikes_Th, che costituisce la soglia per attribuire lo stato di AlmostFull o lo stato di AlmostEmpty. I valori scelti per questo parametro sono compresi tra 1 a 10.

Nella figura 4.18 vi è una leggera diminuzione delle prestazioni con l'aumentare del valore di numBikes_Th, mentre nella figura 4.19 è presente in dettaglio l'andamento dell'accuratezza.

### 4.3.3 Caso TimeslotClassifier

In questo punto, si andranno a effettuare le stesse analisi eseguite precedentemente, ma considerando la classe TimeslotClassifier (paragrafo 3.7).

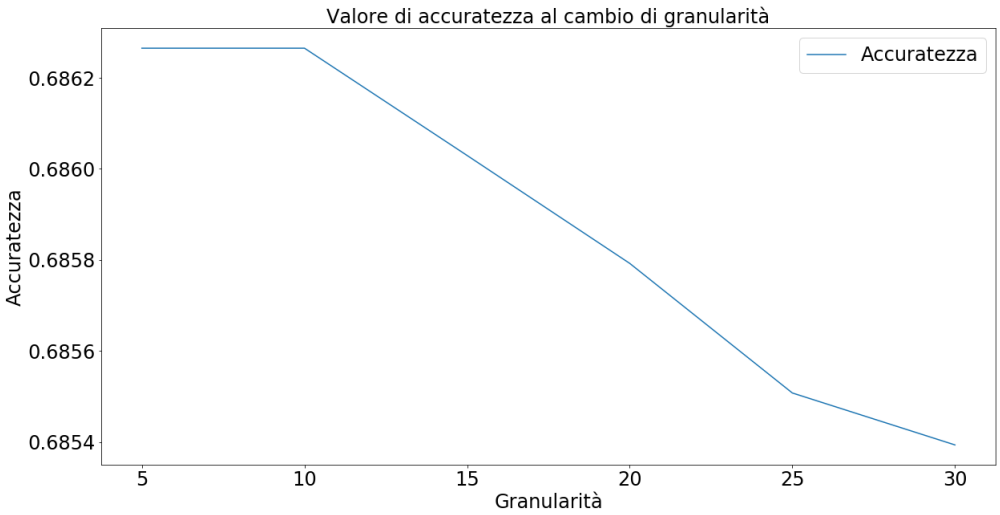


*Figura 4.17: Andamento dell'accuratezza al variare della granularità, nel caso delle prime dieci stazioni.*

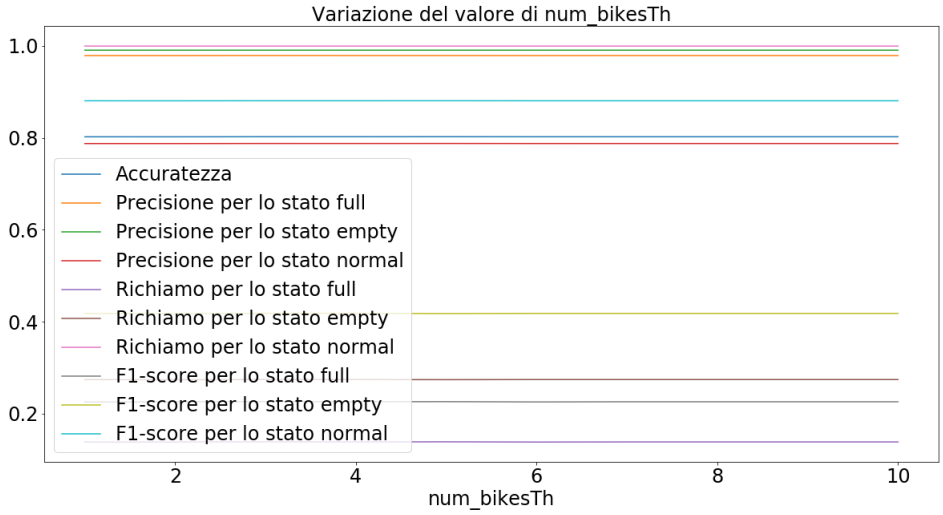


*Figura 4.18: Variazione del valore di numBikes_Th, nel caso delle prime dieci stazioni.*

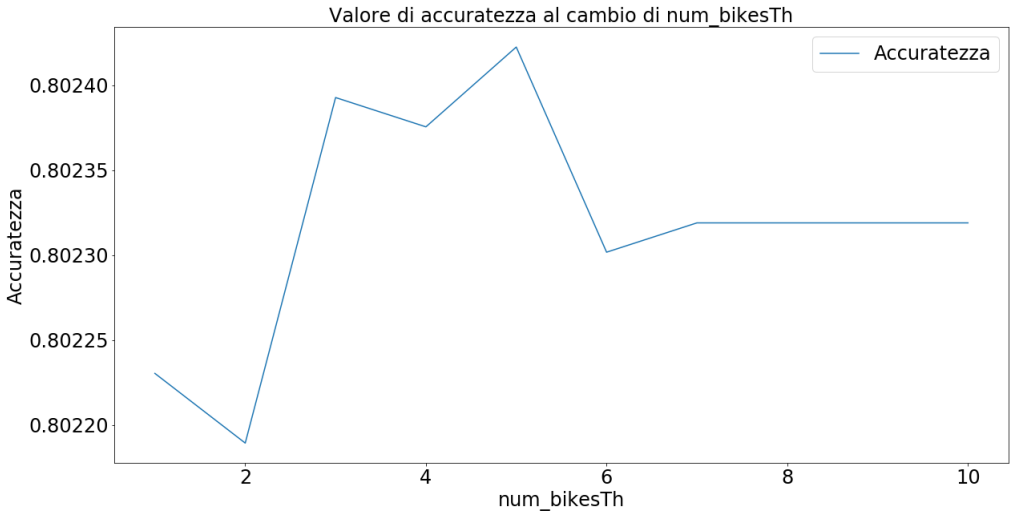


*Figura 4.19: Andamento dell'accuratezza al variare di numBikes_Th, nel caso delle prime dieci stazioni.*

#### Variazione della granularità

La variazione della granularità e i parametri considerati sono gli stessi del paragrafo precedente. Nella figura 4.20 viene riportato l'andamento delle misure relative a questa tipologia di esperimento: anche esso è decrescente, ma con la differenza che si ottengono dei valori più alti del caso precedente. Nella figura 4.21 è possibile vedere come vari l'accuratezza all'aumentare del valore di granularità.

#### Variazione del valore di numBikes_Th

Per quanto riguarda la variazione del valore di numBikes_Th, si tiene sempre fisso il parametro di granularità a 5 minuti e si considera lo stesso range di valori di numBikes_Th.

Nella figura 4.22, l'andamento ottenuto mostra sempre dei miglioramenti rispetto alla classe Classifier. La curva presente nella figura 4.23 rappresenta l'accuratezza al variare del valore di numBikes_Th.

### 4.3.4 Variazione della granularità e variazione numBikes_Th sulle prime dieci stazioni (in dettaglio)

Possiamo ripetere la stessa tipologia di esperimento mostrando in dettaglio la situazione sulle misure delle prime dieci stazioni.

Nel caso della predizione degli stati Full, Empty e Normal si variano i parametri di granularità pari a 10 e a 20 minuti, tenendo fissi gli altri parametri di classificazione, con gli stessi valori degli esperimenti precedenti.

Nelle figure 4.24 e 4.25 vengono riportati i grafici relativi alle prime dieci stazioni, in cui è possibile visionare le variazioni del cambio di granularità. Come si può notare, non c'è un sostanziale cambiamento di prestazioni sulle stazioni prese in esame.

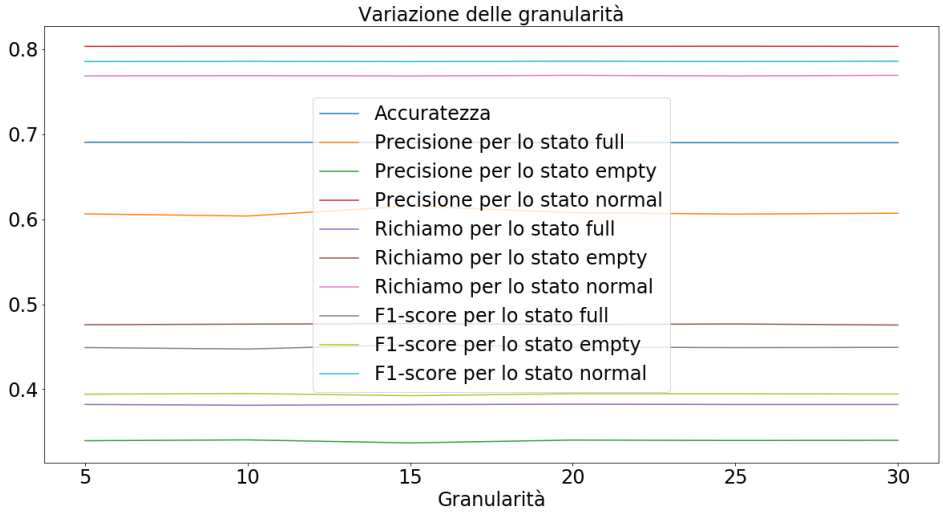


*Figura 4.20: Variazione della granularità nel caso della classificazione basata sulle fasce orarie, nel caso delle prime dieci stazioni.*

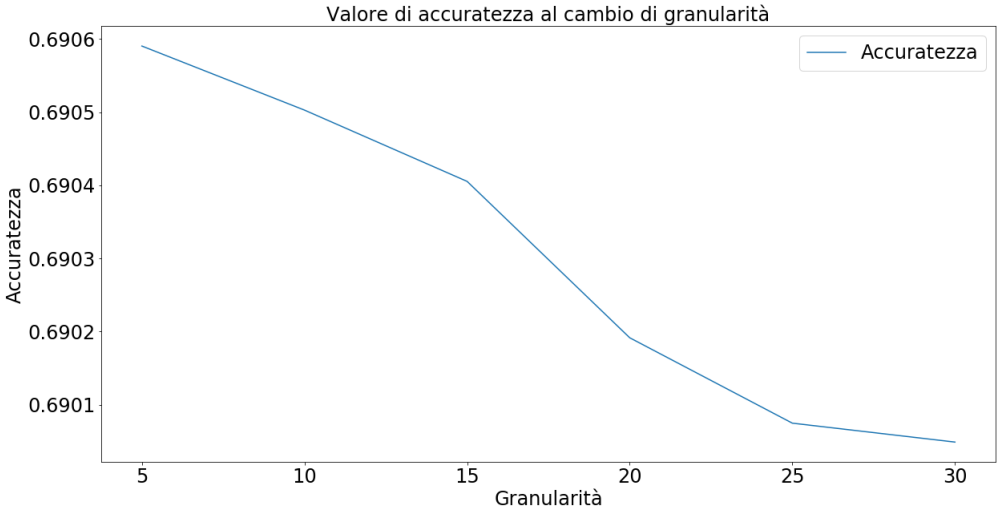


*Figura 4.21: Andamento dell'accuratezza al variare della granularità nel caso della classificazione basata sulle fasce orarie, per le prime dieci stazioni.*

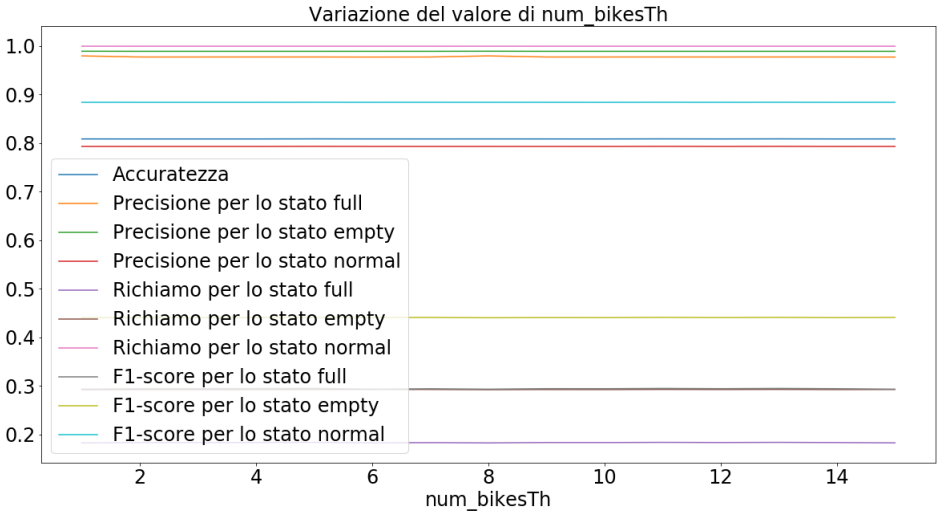


*Figura 4.22: Variazione del valore di numBikes_Th nel caso della classificazione basata sulle fasce orarie, nel caso delle prime dieci stazioni.*

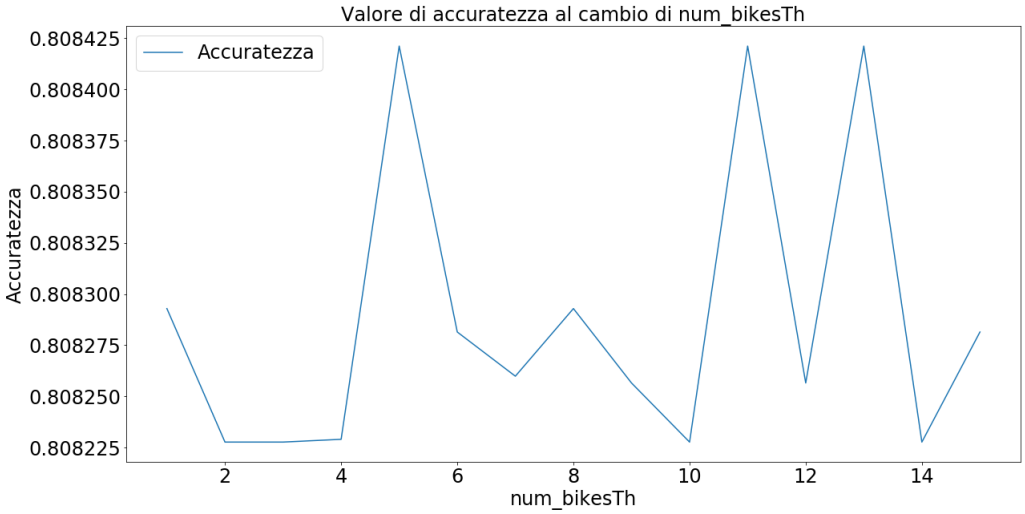


*Figura 4.23: Andamento dell'accuratezza al variare di numBikes_Th nel caso della classificazione basata sulle fasce orarie, per le prime dieci stazioni.*

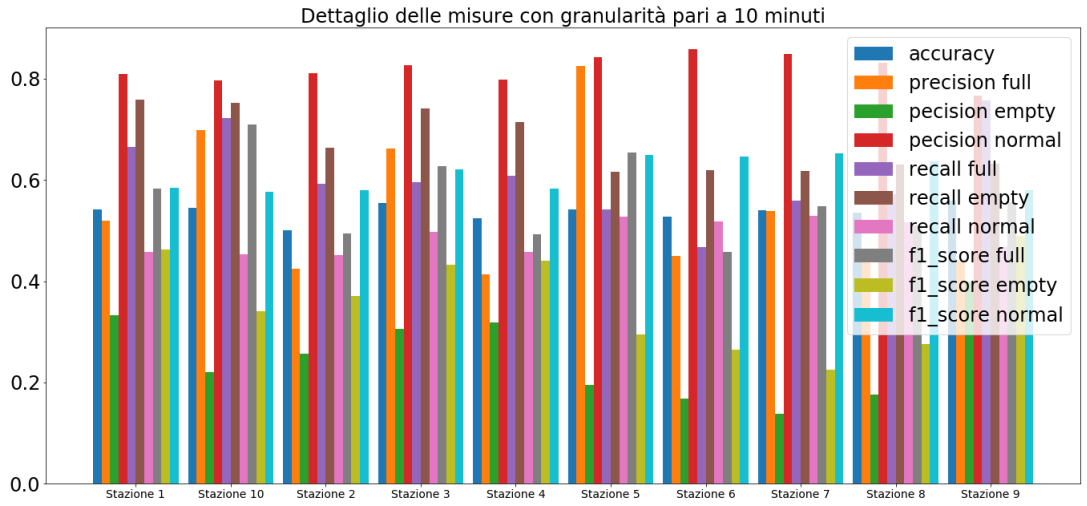


*Figura 4.24: Dettaglio delle misure con granularità pari a 10 minuti nel caso delle prime dieci stazioni prese in esame.*

Considerando il caso degli stati Full, Empty, Normal, AlmostFull e AlmostEmpty utilizzando come valori di soglia 2 e 4 e ripetendo l'analisi, otteniamo i risultati delle figure 4.26 e 4.27, notando come essi rimangano invariati. Tuttavia, rispetto alle precedenti 4.24 e 4.25 si ottiene un miglioramento delle prestazioni, ma parallelamente vi è un peggioramento nel valore di richiamo per gli stati Full e Empty, con un notevole aumento nello stato Normal.

## 4.4 Grid Search dei parametri

### 4.4.1 Ricerca dei parametri per il caso del Classifier

Il primo esperimento che si può svolgere sulla classificazione lo si fa considerando solamente le stazioni aventi come stato:

- Full;
- Empty;
- Normal.

In questa analisi si generano sequenze aventi la granularità pari a:

- 5 minuti;
- 15 minuti;
- 30 minuti.

Nell'analisi si può svolgere un grid search sui parametri della classe Classifier, utilizzando i seguenti valori:

- MinSupp: con valori pari a 0.01, 0.001 e 0.0001;

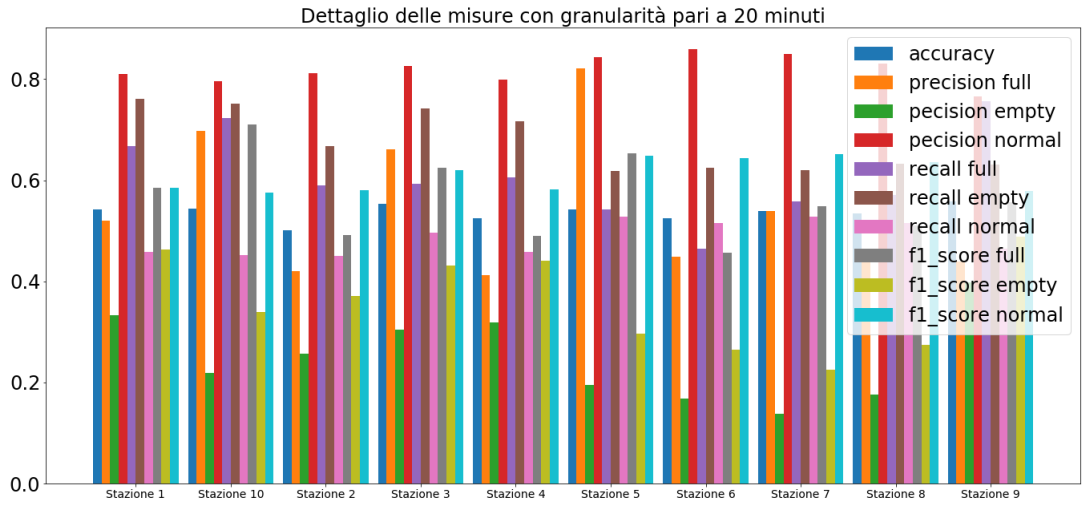


*Figura 4.25: Dettaglio delle misure con granularità pari a 20 minuti nel caso delle prime dieci stazioni prese in esame.*

- K: con valori pari a 1, 2, 3 e 4;
- type_prediction: con valori di weighted e majority;
- K_restriction: con valori pari a true e false.

Nel caso di granularità 5 minuti, le prestazioni migliori si ottengono con i valori di minimo supporto pari a 0.0001 e con un valore di K uguale a 1, che portano ad ottenere un valore di accuratezza medio (su tutte le stazioni) pari a 0.7978. Notiamo, tenendo fisso il valore di minimo supporto, che con l'aumentare del valore di K, le prestazioni peggiorano di poco.

Il valore di type_prediction non comporta una sostanziale variazione dei risultati. Invece, il parametro di K_restriction con valore uguale a false assume valori maggiori con l'aumentare dei valori di minimo supporto.

Nella tabella 4.1 vengono mostrati i risultati migliori corrispondenti ai valori di granularità pari a 5 minuti e minimo supporto pari a 0.0001.

Come è possibile notare dal grafico (figura 4.28) relativo al riassunto delle misure medie sulle stazioni, in base alle predizioni degli stati si può notare che le prestazioni relative ad accuratezza, precisione e richiamo decrescono con l'aumentare della granularità. La figura 4.29 rappresenta la variazione dell'accuratezza all'aumentare del valore di granularità.

La figura 4.30 rappresenta l'andamento delle misure al variare del minimo supporto, con parametri uguali allo studio della figura precedente (4.29). Inoltre, nella figura 4.31 è possibile visionare il dettaglio dell'andamento dell' accuratezza.

Per la configurazione in questione i migliori parametri sono:

- Minimo supporto: 0.0001;
- K: 1;
- type_prediction: weighted;

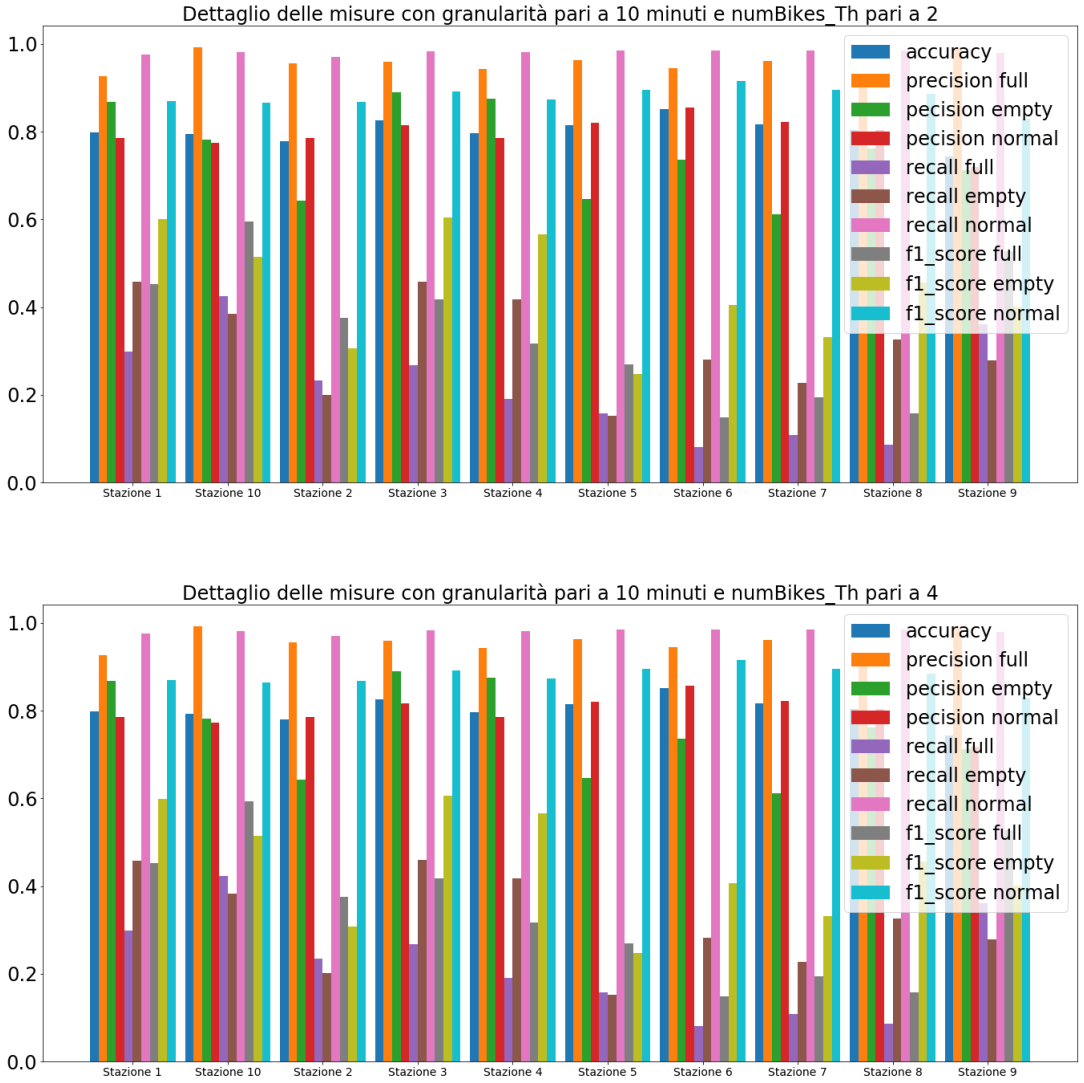


*Figura 4.26: Dettaglio delle misure con granularità pari a 10 minuti e con valore di numBikes_Th pari a 2 e a 4, nel caso delle prime dieci stazioni prese in esame.*

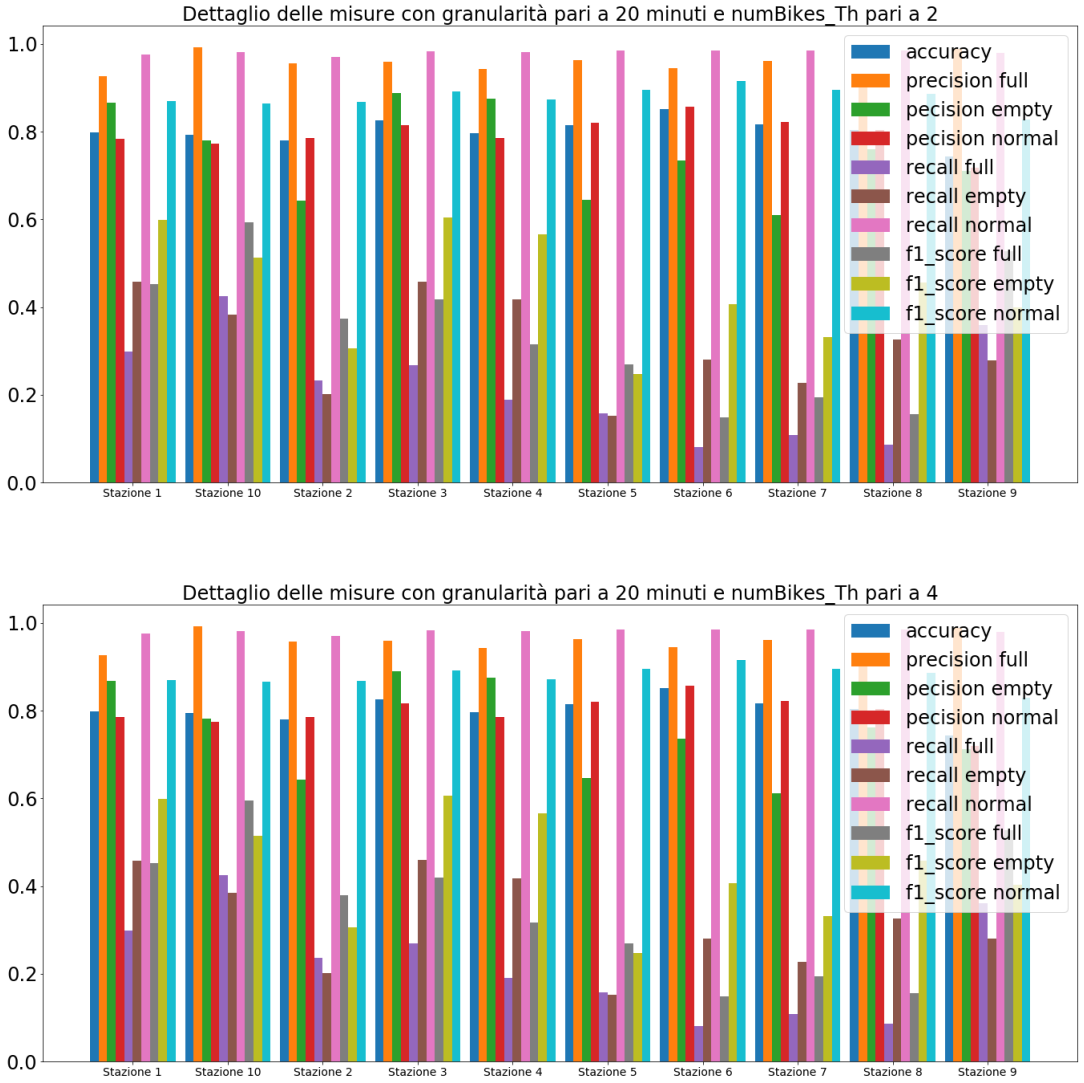


*Figura 4.27: Dettaglio delle misure con granularità pari a 20 minuti e con valore di numBikes_Th pari a 2 e a 4, nel caso delle prime dieci stazioni prese in esame.*

| minSupp | k | type_prediction | K_restriction | accuracy_avg |
|---|---|---|---|---|
| 0.0001 | 1 | weighted | T | 0.7978 |
| 0.0001 | 1 | weighted | F | 0.7978 |
| 0.0001 | 1 | majority | T | 0.7978 |
| 0.0001 | 1 | majority | F | 0.7978 |
| 0.0001 | 2 | weighted | T | 0.7977 |
| 0.0001 | 2 | weighted | F | 0.7977 |
| 0.0001 | 2 | majority | T | 0.7977 |
| 0.0001 | 2 | majority | F | 0.7977 |
| 0.0001 | 3 | weighted | T | 0.7959 |
| 0.0001 | 3 | weighted | F | 0.7977 |
| 0.0001 | 3 | majority | T | 0.7959 |
| 0.0001 | 3 | majority | F | 0.7977 |
| 0.0001 | 4 | weighted | T | 0.7480 |
| 0.0001 | 4 | weighted | F | 0.7977 |
| 0.0001 | 4 | majority | T | 0.7480 |
| 0.0001 | 4 | majority | F | 0.7977 |

*Tabella 4.1: Risultati derivati al grid search della classe Classifier con la restrizione di rappresentazione del solo minimo supporto pari 0.0001 e granularità pari a 5.*

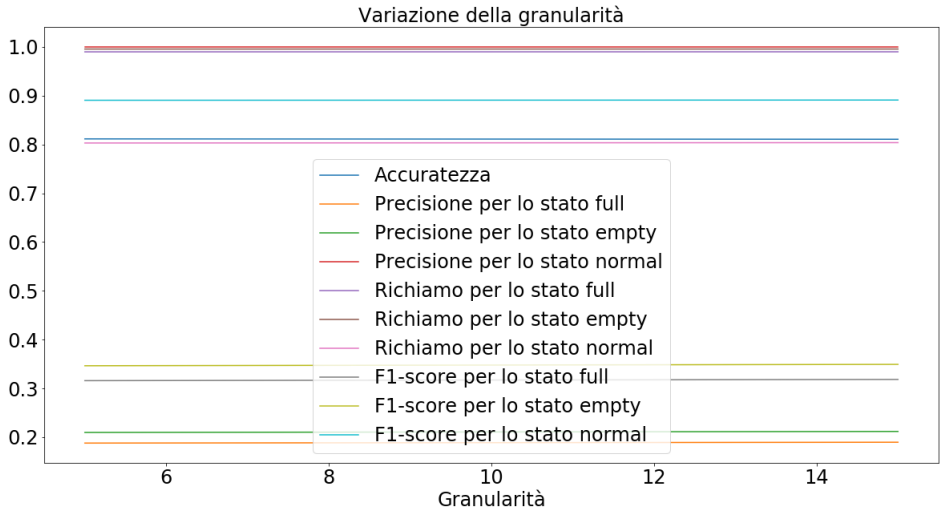


*Figura 4.28: Dettaglio delle misure medie su tutte le stazioni, con minimo supporto pari a 0.0001, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare della granularità.*

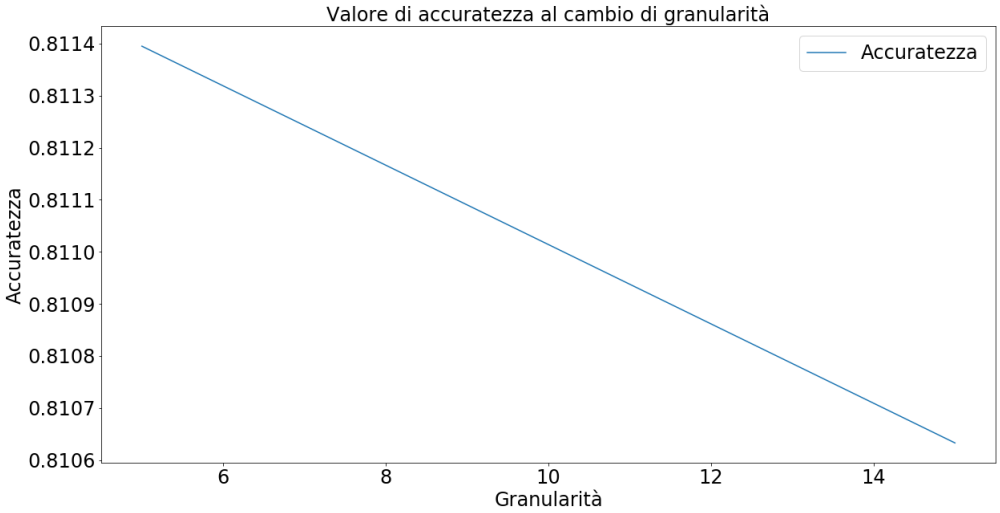


*Figura 4.29: Dettaglio della misura di accuratezza media su tutte le stazioni, con minimo supporto pari a 0.0001, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare della granularità.*

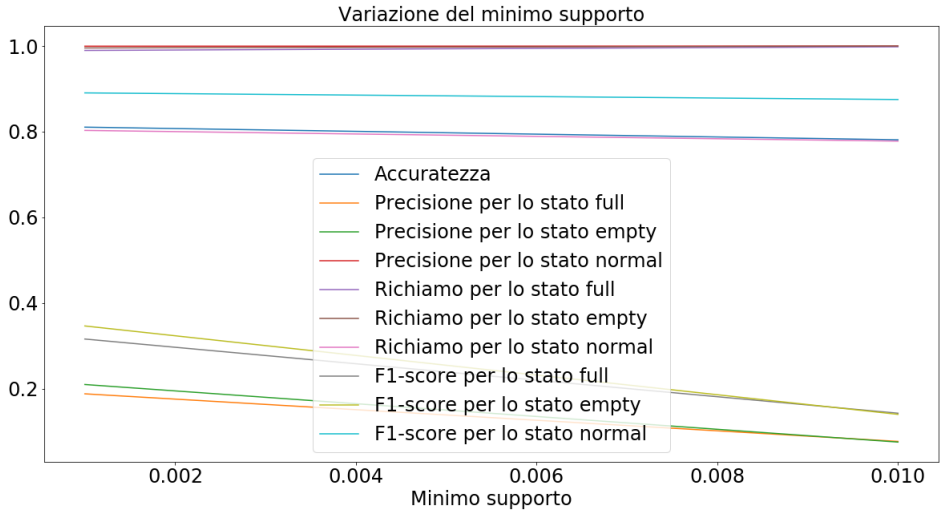


*Figura 4.30: Dettaglio delle misure con granularità pari a 5, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare del minimo supporto.*

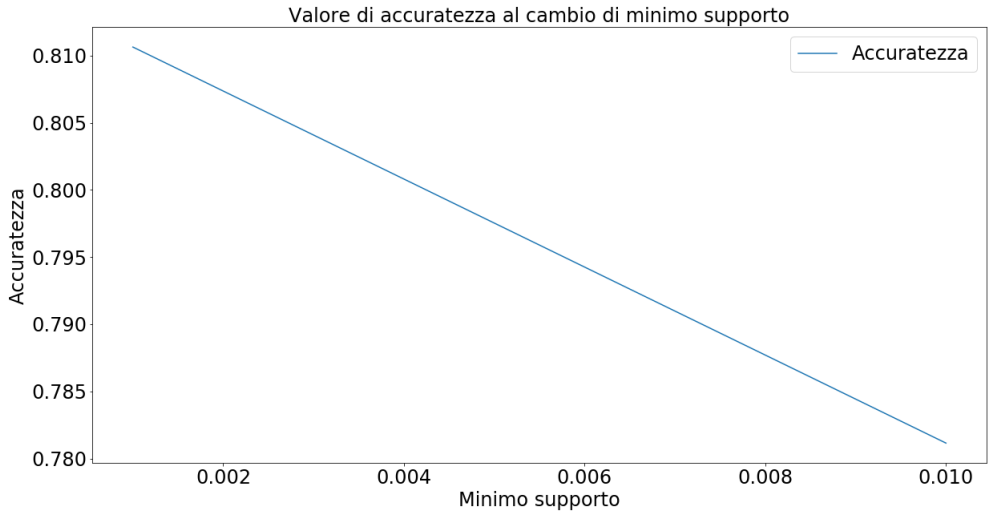


*Figura 4.31: Dettaglio della misura di accuratezza con granularità pari a 5, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare del minimo supporto.*

| | | **Stati predetti** | | |
|---|---|---|---|---|
| | | Full | Empty | Normal |
| **Stati reali** | Full | 30655 | 38 | 137889 |
| | Empty | 84 | 149493 | 566369 |
| | Normal | 322 | 1978 | 2588169 |

*Tabella 4.2: Matrice di confusione della classe Classifier, con parametri: granularità pari a 5 minuti, minimo supporto pari a 0.0001, K pari a 1, type_prediction pari a weighted e K_restriction pari a true.*

- K_restriction: true;

si può rappresentare la matrice di confusione, presente nella tabella 4.2, con accuratezza finale è pari a 0.7948.

### 4.4.2 Ricerca dei parametri per il caso del Classifier nella versione con gli stati AlmostFull e AlmostEmpty

In questo caso si prende in esame sempre la ricerca degli hyperparameters della classe Classifier, ma con l'aggiunta degli stati di AlmostFull e AlmostEmpty.

Il dataset viene modificato con i seguenti valori:

- granularità: 5, 15 e 30 minuti;
- numBikes_Th: 2 e 4.

Per l'analisi si prendono in considerazione i seguenti valori:

- minSupp: con valori pari a 0.01, 0.001 e 0.0001;
- K: con valori pari a 1, 2, 3 e 4;

| minSupp | k | type_prediction | K_restriction | accuracy_avg |
|---|---|---|---|---|
| 0.0001 | 1 | weighted | F | 0.8184 |
| 0.0001 | 1 | weighted | T | 0.8183 |
| 0.0001 | 1 | majority | F | 0.8182 |
| 0.0001 | 1 | majority | T | 0.8184 |
| 0.0001 | 2 | weighted | F | 0.7983 |
| 0.0001 | 2 | weighted | T | 0.7983 |
| 0.0001 | 2 | majority | F | 0.7984 |
| 0.0001 | 2 | majority | T | 0.7982 |
| 0.0001 | 3 | weighted | F | 0.7876 |
| 0.0001 | 3 | weighted | T | 0.7868 |
| 0.0001 | 3 | majority | F | 0.7921 |
| 0.0001 | 3 | majority | T | 0.7868 |
| 0.0001 | 4 | weighted | F | 0.7984 |
| 0.0001 | 4 | weighted | T | 0.7868 |
| 0.0001 | 4 | majority | F | 0.7834 |
| 0.0001 | 4 | majority | T | 0.7935 |

*Tabella 4.3: Risultati derivati al grid search della classe Classifier con l'aggiunta dei casi di AlmostFull e AlmostEmpty, con la restrizione di rappresentazione del solo minimo supporto pari a 0.0001, ganularità pari a 5 e numBikes_Th pari a 4.*

- type_prediction: con valori di weighted e majority;
- K_restriction: con valori pari a true e false.

I risultati migliori ottenuti in questa analisi sono:

- Granularità: 5 minuti;
- Minimo supporto: 0.0001;
- numBikes_Th: 4.

Nella tabella 4.3 è possibile visionare i risultati migliori aventi questi valori, facendo variare gli altri valori, cioè K, type_prediction e K_restriction.

Nella figura 4.32 è possibile vedere l'andamento delle curve delle misure al variare del valore di granularità, mentre nella figura 4.33 vi è rappresentata solamente la misura di accuratezza media su tutte le stazioni per poterla vedere in dettaglio. Ciò che si evince da questi grafici 4.32 e 4.33 è che con l'aumentare del parametro di granularità le prestazioni diminuiscono, anche per il caso del classificatore che utilizza gli stati aggiuntivi AlmostFull e AlmostEmpty.

Nella figura 4.34 sono rappresentate le misure in gioco al variare del minimo supporto e nella figura 4.35 vi è il dettaglio dell'andamento della misura di accuratezza. Al variare del minimo supporto si ottiene una diminuzione delle prestazioni, come nel caso di classificazione senza gli stati di AmostFull e AlmostEmpty.

Nella figura 4.36 possiamo vedere come si comportano le curve con il variare del valore di numBikes_Th, e si può notare dalla figura 4.37 che l'andamento dell'accuratezza cresce con l'aumentare del valore di numBikes_Th.

Per la configurazione in questione i migliori parametri sono:

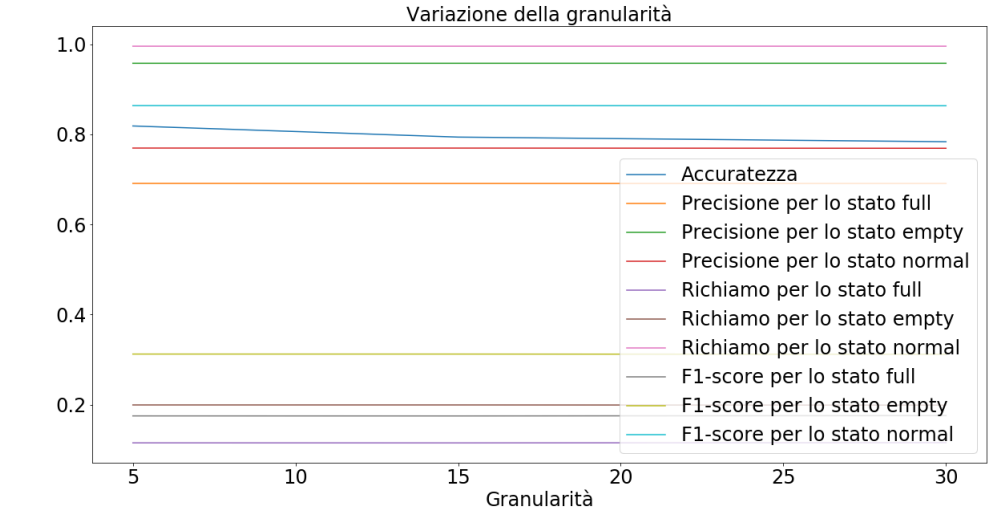


*Figura 4.32: Dettaglio delle misure medie su tutte le stazioni, con minimo supporto pari a 0.0001, K pari a 1, numBikes_Th pari a 4, type_predicition con valore weighted e K_restriction con valore false al variare della granularità.*

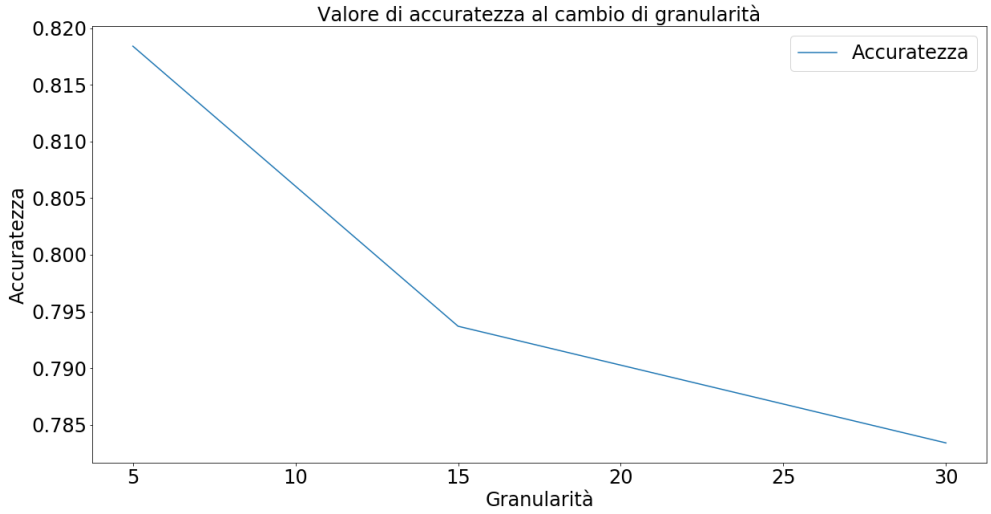


*Figura 4.33: Dettaglio della misura di accuratezza media su tutte le stazioni, con minimo supporto pari a 0.0001, K pari a 1, numBikes_Th pari a 4, type_predicition con valore weighted e K_restriction con valore false al variare della granularità.*

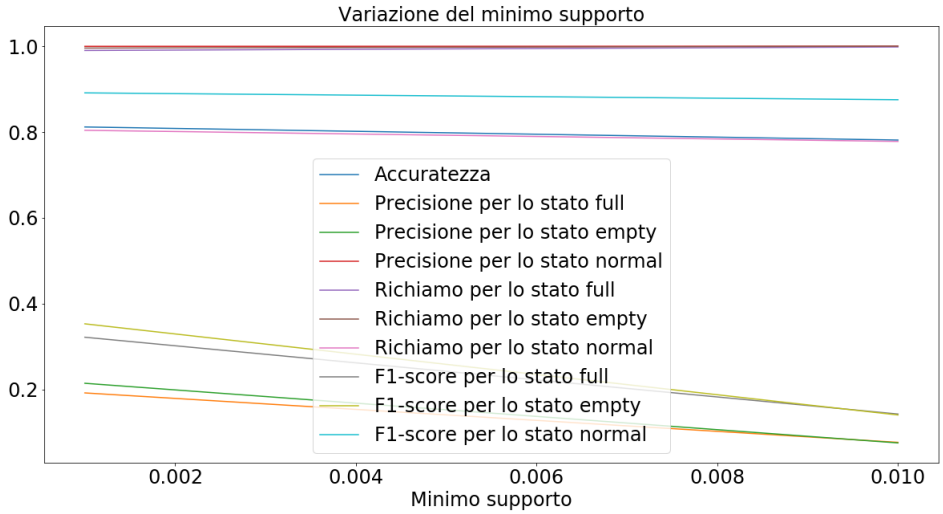


*Figura 4.34: Dettaglio delle misure nella configurazione con granularità pari a 5, numBike_Th pari a 4, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare del minimo supporto.*

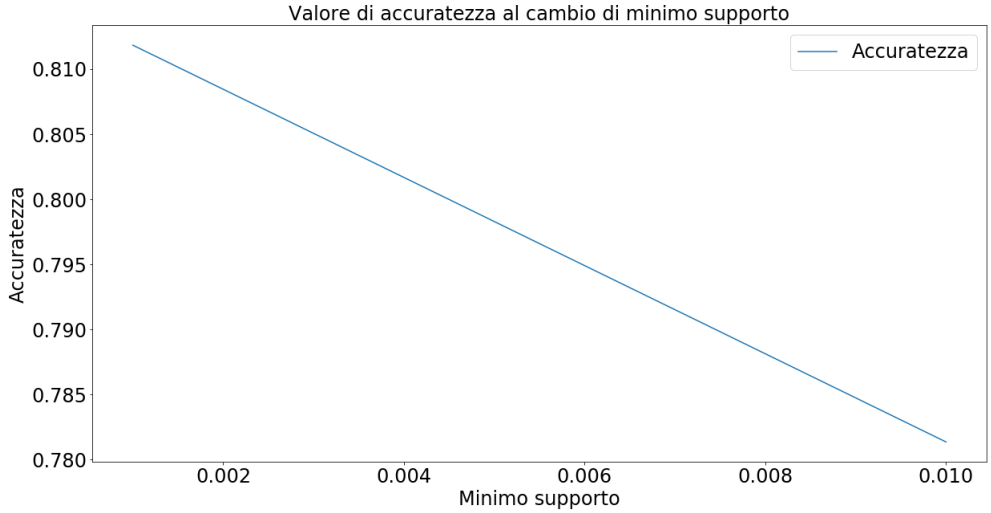


*Figura 4.35: Dettaglio della misura di accuratezza nella configurazione con granularità pari a 5, numBikes_Th pari a 4, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare del minimo supporto.*

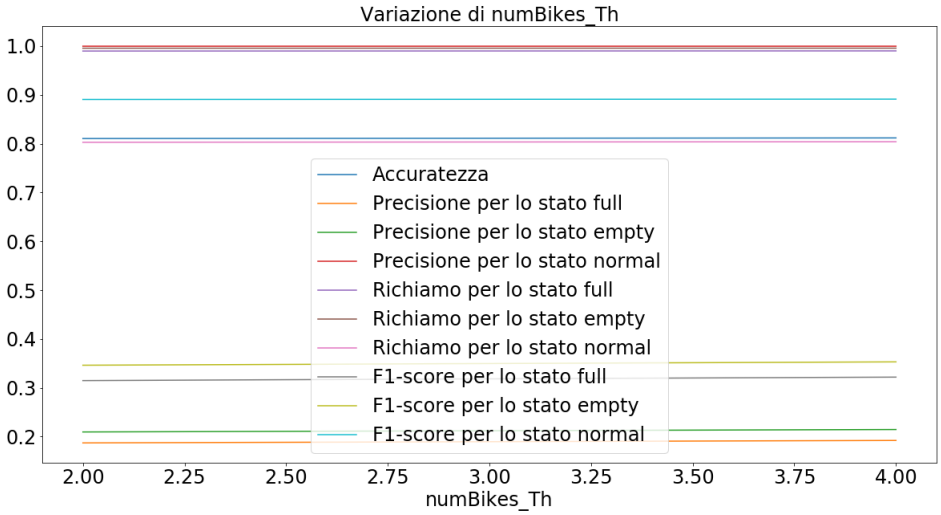


*Figura 4.36: Dettaglio delle misure nella configurazione con granularità pari a 5, minimo supporto pari a 0.0001, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare di numBikes_Th.*

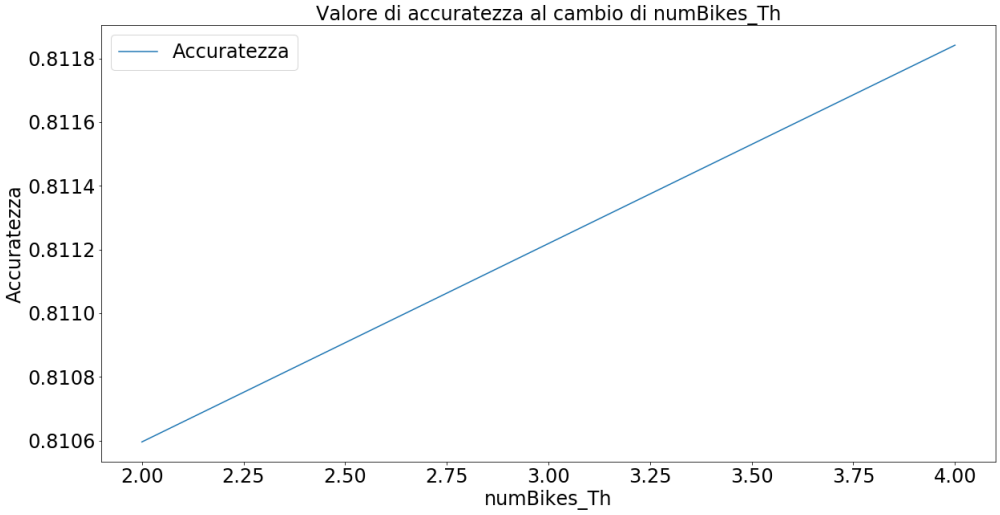


*Figura 4.37: Dettaglio della misura di accuratezza nella configurazione con granularità pari a 5, minimo supporto 0.0001, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare di numBikes_Th.*

| | | Stati predetti | | |
|---|---|---|---|---|
| | | Full | Empty | Normal |
| **Stati reali** | Full | 331176 | 37 | 135897 |
| | Empty | 57 | 146819 | 568324 |
| | Normal | 309 | 1963 | 2389454 |

*Tabella 4.4: Matrice di confusione della classe Classifier, con parametri: granularità pari a 5 minuti, minimo supporto pari a 0.0001, numBikes_Th pari a 4, K pari a 1, type_prediction pari a weighted e K_restriction pari a true.*

- Minimo supporto: 0.0001;
- K: 1;
- type_prediction: weighted;
- K_restriction: true;
- numBikes_th: 4.

Si può rappresentare la matrice di confusione, presente nella tabella 4.4, in cui si ottiene come accuratezza globale 0.8023.

### 4.4.3 Ricerca dei parametri per il caso della classe TimeslotClassifier

In questo parte del capitolo si sottopone la classe TimeslotClassifier alla ricerca dei suoi valori di hyperparameters, utilizzando sempre la tecnica del grid search, mantenendo fisso il set di fasce orarie, cioè:

- dalle ore 4:00 alle ore 9:59;
- dalle ore 10:00 alle ore 15:59;
- dalle ore 16:00 alle ore 21:59;
- dalle ore 22:00 alle ore 3:59.

In questo caso si utilizzeranno i seguenti parametri per effettuare la ricerca:

- Minimo supporto: con valori pari a 0.01, 0.001;
- Granularità: con valori pari a 5 e 15 minuti;
- K: con valori pari a 1, 2, 3 e 4;
- type_prediction: con valori di weighted e majority;
- K_restriction: con valori pari a true e false.

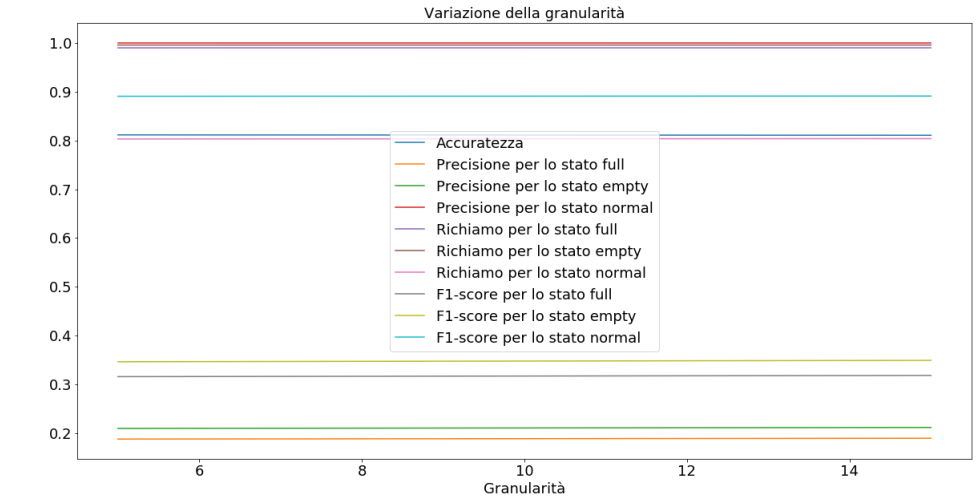


*Figura 4.38: Dettaglio delle misure medie su tutte le stazioni utilizzando la classe TimeslotClassifier, nella configurazione con minimo supporto pari a 0.001, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare della granularità.*

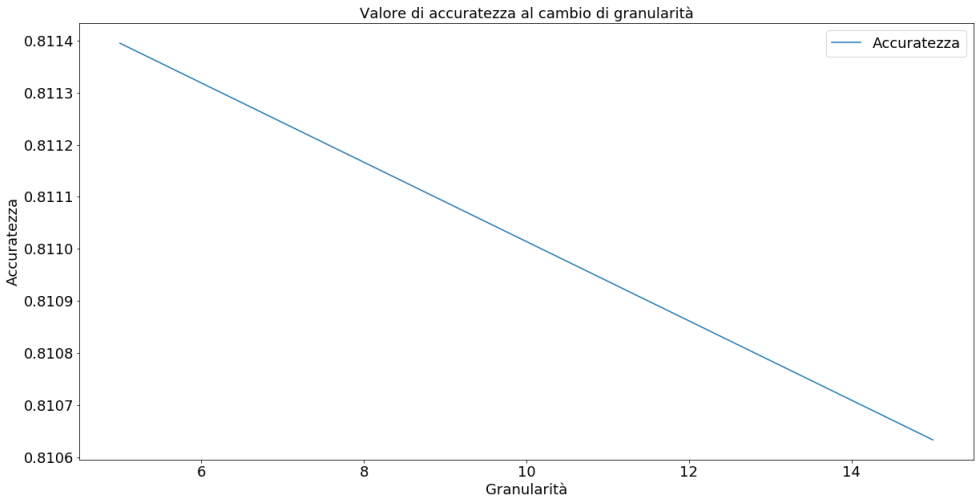


*Figura 4.39: Dettaglio della misura di accuratezza media su tutte le stazioni utilizzando la classe TimeslotClassifier, nella configurazione con minimo supporto pari a 0.001, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare della granularità.*

| minSupp | k | type_prediction | K_restriction | accuracy_avg |
|---|---|---|---|---|
| 0.001 | 1 | weighted | F | 0.8114 |
| 0.001 | 1 | weighted | T | 0.8114 |
| 0.001 | 1 | majority | F | 0.8114 |
| 0.001 | 1 | majority | T | 0.8114 |
| 0.001 | 2 | weighted | F | 0.8106 |
| 0.001 | 2 | weighted | T | 0.8110 |
| 0.001 | 2 | majority | F | 0.8106 |
| 0.001 | 2 | majority | T | 0.8106 |
| 0.001 | 3 | weighted | F | 0.8109 |
| 0.001 | 3 | weighted | T | 0.8113 |
| 0.001 | 3 | majority | F | 0.8106 |
| 0.001 | 3 | majority | T | 0.8112 |
| 0.001 | 4 | weighted | F | 0.8106 |
| 0.001 | 4 | weighted | T | 0.8110 |
| 0.001 | 4 | majority | F | 0.8014 |
| 0.001 | 4 | majority | T | 0.8113 |

*Tabella 4.5: Risultati derivati dal grid search della classe TimeslotClassifier, con la restrizione di rappresentazione del solo minimo supporto pari a 0.001 e granularità pari a 5 minuti.*

Nella figura 4.38 è possibile vedere il comportamento delle misure al variare della granularità, nella figura 4.39 si può vedere come l'accuratezza media sulle stazioni varia.

Nella tabella 4.5 è possibile vedere i risultati relativi al valore di granularità pari a 5 minuti e di minimo supporto pari a 0.001: come si può notare si ottengono dei valori maggiori rispetto al caso della classificazione basata sulla classe Classifier (tabella 4.1).

Possiamo considerare la seguente configurazione:

- Minimo supporto: 0.001;
- K: 1;
- type_prediction: weighted;
- K_restriction: true;

si può rappresentare la matrice di confusione, presente nella tabella 4.6, con accuratezza globale di 0.8114.

### 4.4.4 Ricerca dei parametri per il caso della classe TimeslotClassifier nella versione con gli stati AlmostFull e AlmostEmpty

Come svolto per la classe Classifier, in questo punto, utilizzando la classe TimeslotClassifier si estenderà l'analisi del grid search introducendo gli stati di AlmostFull e AlmostEmpty. L'analisi del grid search verterà sull'uso dei seguenti range di parametri:

| | | Stati predetti | | |
|---|---|---|---|---|
| | | Full | Empty | Normal |
| **Stati reali** | Full | 27512 | 0 | 117532 |
| | Empty | 96 | 122708 | 456676 |
| | Normal | 184 | 540 | 2351112 |

*Tabella 4.6: Matrice di confusione della classe TimeslotClassifier, con parametri: granularità pari a 5 minuti, minimo supporto pari a 0.001, K pari a 1, type_prediction pari a weighted e K_restriction uguale a true.*

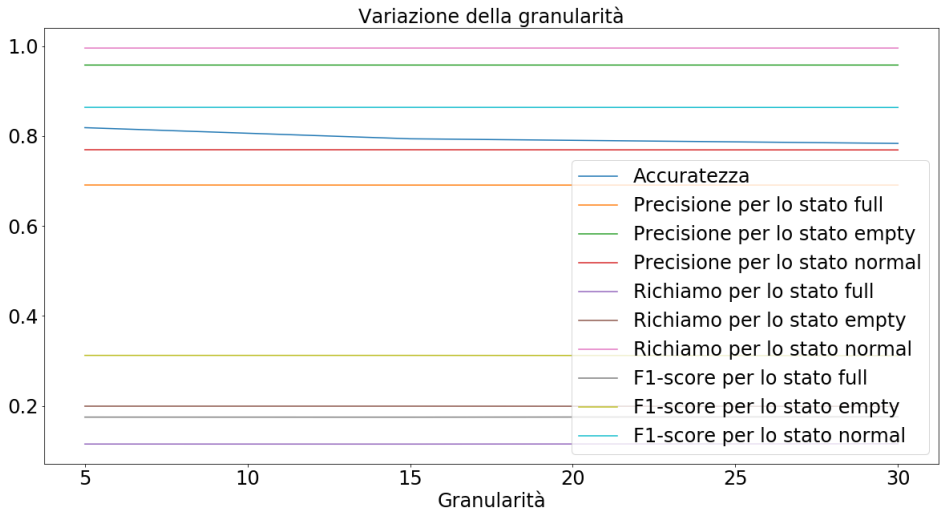


*Figura 4.40: Dettaglio delle misure medie su tutte le stazioni utilizzando la classe TimeslotClassifier, nella configurazione con minimo supporto pari a 0.001, numBikes_Th pari a 4, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare della granularità.*

- Minimo supporto: 0.01 e 0.001;
- Granularità: 5 e 15 minuti;
- numBikes_Th: 2 e 4;
- K: 1, 2, 3 e 4;
- type_prediction: con valori di weighted e majority;
- K_restriction: con valori pari a true e false.

Nel grafico 4.40 è possibile vedere l'andamento delle misure medie sulle stazioni, considerando la variazione di granularità, mentre nel grafico 4.41 si può vedere come l'accuratezza decresca all'aumentare del valore di granularità.

Nella figura 4.42 possiamo visionare, invece, il valore di minimo supporto e valutare come vengono trattate le misure medie sulle stazioni, mentre nella figura 4.43 vi è rappresentato l'andamento decrescente dell'accuratezza al variare del minimo supporto.

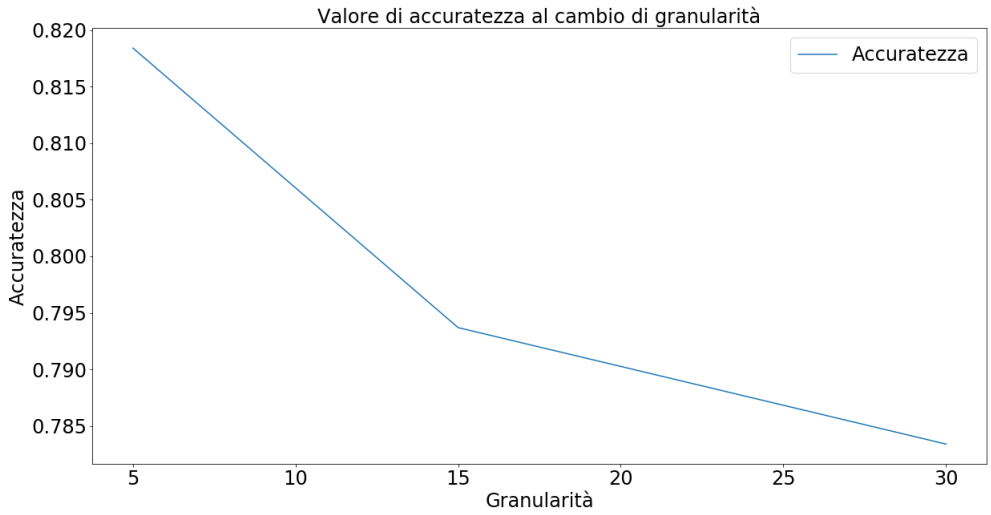


*Figura 4.41: Dettaglio della misura di accuratezza media su tutte le stazioni utilizzando la classe TimeslotClassifier, nella configurazione con minimo supporto pari a 0.001, numBikes_Th pari a 4, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare della granularità.*

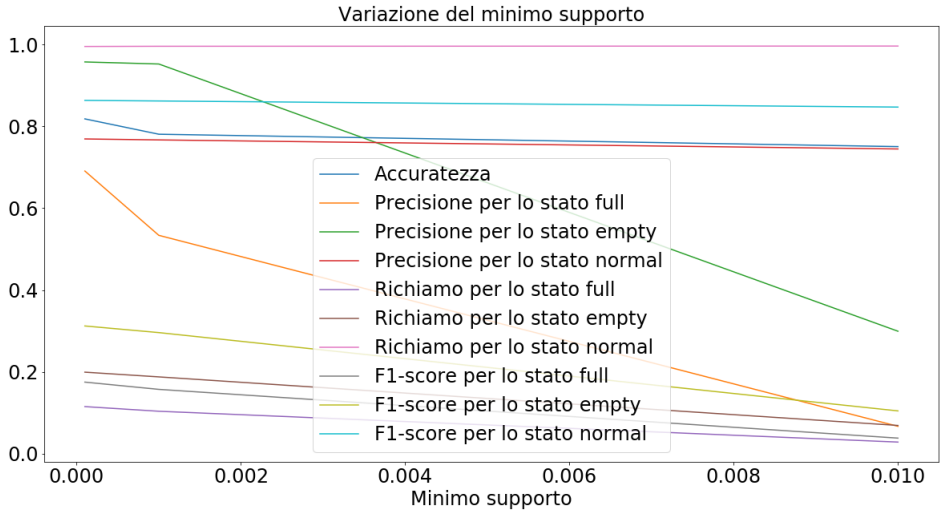


*Figura 4.42: Dettaglio delle misure utilizzando la classe TimeslotClassifier, nella configurazione con granularità pari a 5, numBike_Th pari a 4, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare del minimo supporto.*

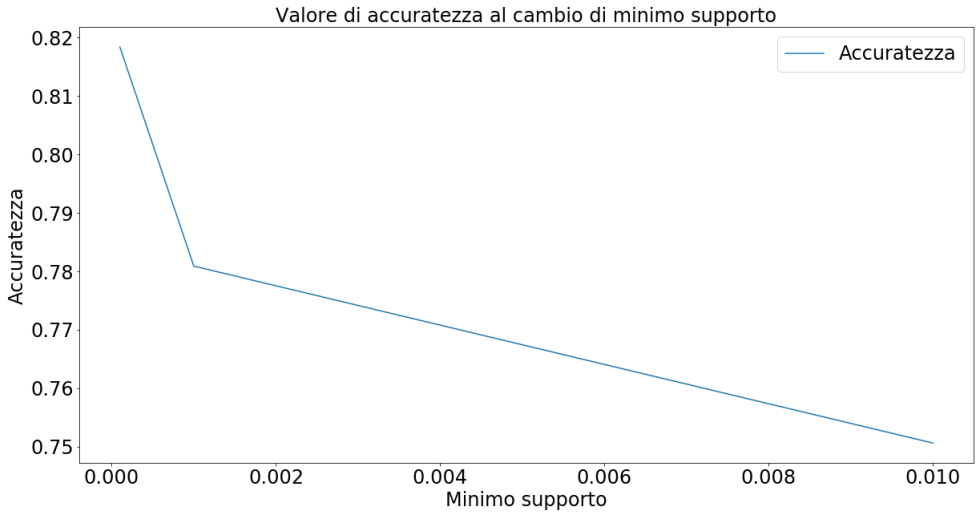


*Figura 4.43: Dettaglio della misura di accuratezza utilizzando la classe TimeslotClassifier, nella configurazione con granularità pari a 5, numBikes_Th pari a 4, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare del minimo supporto.*

Nella figura 4.44 possiamo vedere come si comportano le curve con il valore di numBikes_Th, e si può notare, nel dettaglio, dalla figura 4.45, che l'andamento dell'accuratezza cresce con l'aumentare del valore di numBikes_Th.

Nella tabella 4.7 è possibile vedere i risultati relativi al valore di granularità pari a 5 minuti, minimo supporto pari a 0.001 e il valore di numBikes_Th pari a 4.

Possiamo considerare la seguente configurazione:

- Minimo supporto: 0.001;
- K: 1;
- type_prediction: weighted;
- K_restriction: true;
- numBikes_th: 4;

si può rappresentare la matrice di confusione, presente nella tabella 4.8, con accuratezza globale di 0.8118.

## 4.5 Analisi sui possibili valori di fasce orarie

Un'altra analisi su cui possiamo soffermarci è quella relativa al valore di fasce orarie da utilizzare all'interno della classe TimeslotClassifer. In questa punto del capitolo, analizzeremo solamente il caso di sequenze generate con gli stati:

- Full;
- Empty;

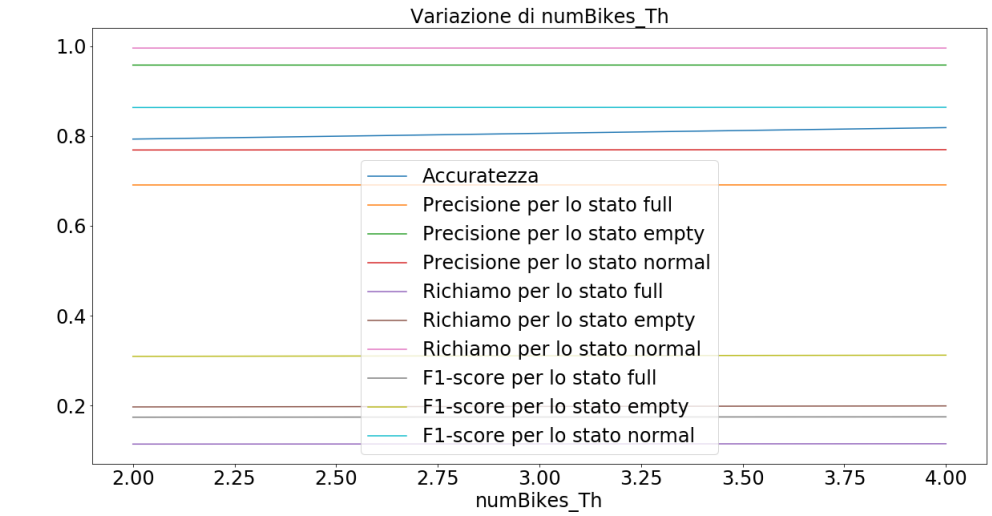


*Figura 4.44: Dettaglio delle misure utilizzando la classe TimeslotClassifier, nella configurazione con granularità pari a 5, minimo supporto pari a 0.001, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare di numBikes_Th.*

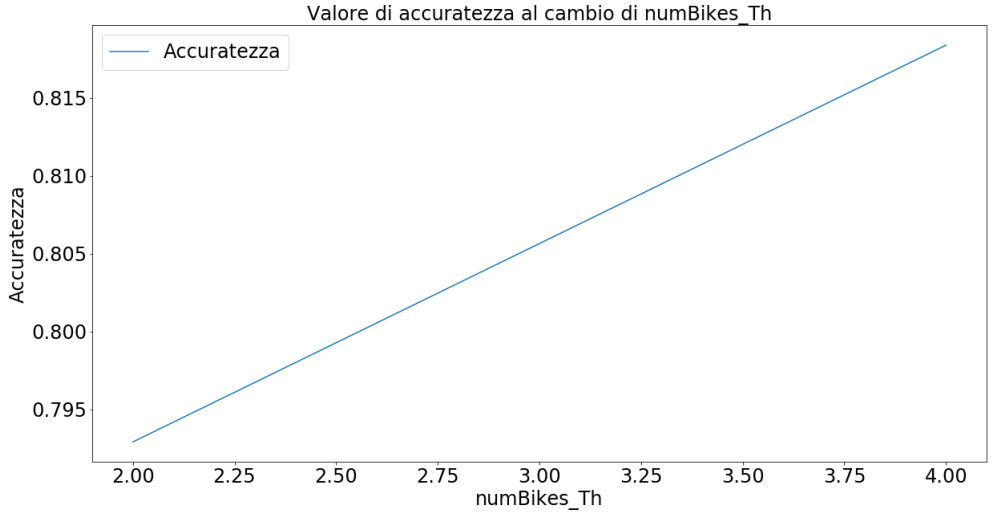


*Figura 4.45: Dettaglio della misura di accuratezza utilizzando la classe TimeslotClassifier, nella configurazione con granularità pari a 5, minimo supporto pari a 0.001, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare di numBikes_Th.*

| minSupp | k | type_prediction | K_restriction | accuracy_avg |
|---|---|---|---|---|
| 0.001 | 1 | weighted | F | 0.8118 |
| 0.001 | 1 | weighted | T | 0.8118 |
| 0.001 | 1 | majority | F | 0.8118 |
| 0.001 | 1 | majority | T | 0.8118 |
| 0.001 | 2 | weighted | F | 0.8014 |
| 0.001 | 2 | weighted | T | 0.8016 |
| 0.001 | 2 | majority | F | 0.8019 |
| 0.001 | 2 | majority | T | 0.8019 |
| 0.001 | 3 | weighted | F | 0.8118 |
| 0.001 | 3 | weighted | T | 0.8118 |
| 0.001 | 3 | majority | F | 0.8117 |
| 0.001 | 3 | majority | T | 0.8106 |
| 0.001 | 4 | weighted | F | 0.8118 |
| 0.001 | 4 | weighted | T | 0.8013 |
| 0.001 | 4 | majority | F | 0.8011 |
| 0.001 | 4 | majority | T | 0.8118 |

*Tabella 4.7: Risultati derivati dal grid search della classe TimeslotClassifier, con la restrizione di rappresentazione del solo minimo supporto pari 0.001, granularità pari 5 minuti e numBikes_Th pari a 4.*

| | | **Stati predetti** | | |
|---|---|---|---|---|
| | | Full | Empty | Normal |
| **Stati reali** | Full | 27920 | 0 | 117324 |
| | Empty | 96 | 124632 | 455872 |
| | Normal | 184 | 540 | 2352056 |

*Tabella 4.8: Matrice di confusione della classe TimeslotClassifier, con parametri: granularità pari 5 minuti, minimo supporto pari a 0.001, numBikes_Th pari a 4, K pari a 1, type_prediction pari a weighted e K_restriction pari a true.*

- Normal;
- AlmostFull;
- AlmostEmpty;

inoltre, si utilizzerà la seguente configurazione per la classificazione:

- Minimo supporto: 0.001;
- K: 1;
- type_prediction: weighted;
- K_restriction: true;
- numBikes_th: 4.

Possiamo iniziare l'analisi considerando due fasce orarie solamente costituite dai seguenti valori:

- dalle ore 4:00 alle ore 15:59;
- dalle ore 16:00 alle ore 3:59;

questo insieme di fasce orarie viene chiamato fasce_orarie_1. Con questa prima analisi si ottiene come valore di accuratezza medio 0.7895.

Invece, considerando otto valori di queste fasce orarie (fasce_orarie_2):

- dalle ore 4:00 alle ore 6:59;
- dalle ore 7:00 alle ore 9:59;
- dalle ore 10:00 alle ore 12:59;
- dalle ore 13:00 alle ore 15:59;
- dalle ore 16:00 alle ore 18:59;
- dalle ore 19:00 alle ore 21:59;
- dalle ore 22:00 alle ore 0:59;
- dalle ore 1:00 alle ore 3:59;

si ottiene come valore di accuratezza medio 0.8520.

Nella figura 4.46 può vedere come si comporta la classe TimeslotClassifier durante la predizione su diversi set di fasce orarie, considerando i casi di: fasce_orarie_1, default e fasce_orarie_2. Nella figura 4.47 si vede il comportamento dell'accuratezza al variare della configurazione.

Dal valore di fasce_orarie_2 si ottiene un aumento delle prestazioni rispetto ai casi studiati in precedenza. Come si può notare, in aggiunta, si ottengono anche dei valori maggiori di richiamo e di precisione delle classi Full ed Empty. Questo vuol dire che una suddivisione accurata delle fasce orarie comporta una migliore predizione sugli stati dei dati di test, poiché data una fascia oraria, altri dati di training non possano influire sulla predizione di altre fasce.

Dalla tabella 4.9 possiamo vedere i valori presenti nella matrice di confusione nel caso di fasce_orarie_2.

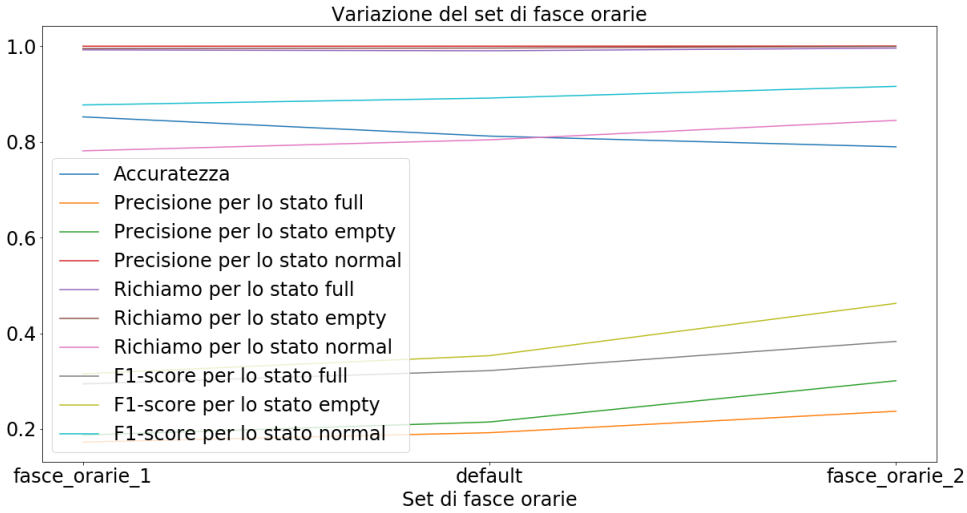


*Figura 4.46: Dettaglio delle misure nella configurazione con granularità pari a 5 minuti, minimo supporto pari a 0.001, numBikes_Th pari a 4, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare della configurazione sul set di fasce orarie.*

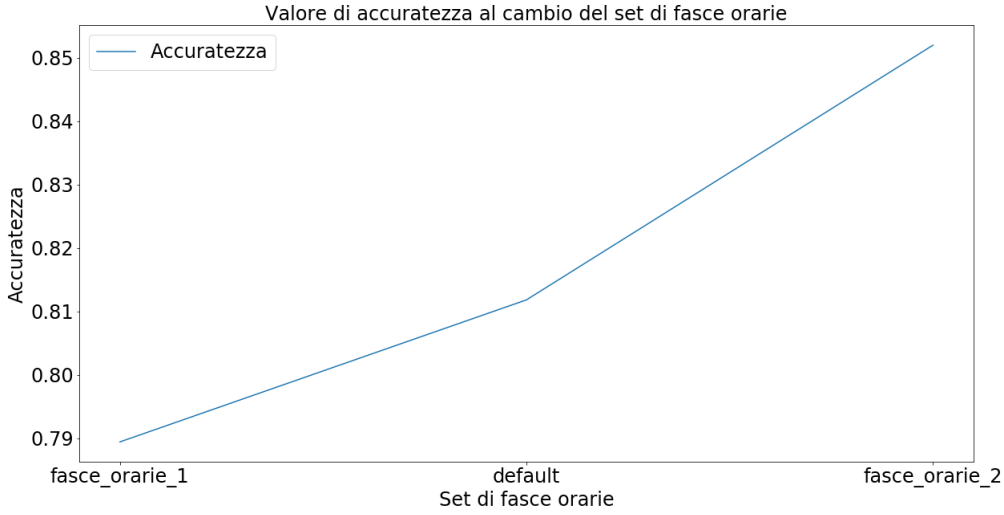


*Figura 4.47: Dettaglio della misura di accuratezza nella configurazione con minimo supporto pari a 5 minuti, minimo supporto pari a 0.001, numBikes_Th pari a 4, K pari a 1, type_predicition con valore weighted e K_restriction con valore false al variare della configurazione sul set di fasce orarie.*

| | | Stati predetti | | |
|---|---|---|---|---|
| | | Full | Empty | Normal |
| **Stati reali** | Full | 32512 | 0 | 104616 |
| | Empty | 96 | 132960 | 308912 |
| | Normal | 48 | 104 | 2248920 |

*Tabella 4.9: Matrice di confusione della classe TimeslotClassifier, con parametri: granularità pari a 5 minuti, minimo supporto pari a 0.001, numBikes_Th pari a 4, K pari a 1, type_prediction pari a weighted e K_restriction pari a true nel caso di fasce_orarie_2.*

## 4.6 Introduzione del parametro di attribuzione degli stati Full ed Empty

Nella nostra analisi può essere utile introdurre un ulteriore parametro per la trasformazione del nostro dataset. Il parametro in questione è chiamato full_empty_th, esso permette di assegnare lo stato di Full e Empty in modo più lasco rispetto alla procedura descritta in precedenza nel paragrafo 3.4.

I risultati ottenuti sulle prime dieci stazioni, insieme al parametro di numBikes_Th, portano ad un peggioramento delle prestazioni, poiché vi sono più stazioni che ricadono negli stati di Full e Empty e ciò comporta anche un aumento del valore di richiamo per queste due classi.

Nella tabella 4.10 viene riportata la configurazione e le misure ottenute durante la classificazione.

| **full_empty_Th** | **numBikes_Th** | **Acc.** | **Prec. Full** | **Prec. Empty** | **Prec. Normal** | **Ric. Full** | **Ric. Empty** | **Ric. Normal** | $F_1$ **Full** | $F_1$ **Empty** | $F_1$ **Normal** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | 0.7245 | 0.9412 | 0.8348 | 0.6916 | 0.2713 | 0.3902 | 0.9714 | 0.4147 | 0.5312 | 0.8070 |
| 2 | 4 | 0.7250 | 0.9412 | 0.8356 | 0.6916 | 0.2718 | 0.3902 | 0.9715 | 0.4153 | 0.5313 | 0.8070 |

*Tabella 4.10: Esperimento sull'introduzione del parametro full_empty_Th.*

# Capitolo 5

# Conclusioni

In questo capitolo verranno esposte le conclusioni e le possibili implementazioni future relative al lavoro di tesi.

## 5.1 Conclusione

Il documento di tesi espone esperimenti e risultati utili per verificare che un classificatore associativo sia in grado di predire situazioni derivanti da condizioni temporali, utilizzando tecniche di sviluppo basate sul framework di Spark, per la gestione di grandi moli di dati.

Inoltre, nei capitoli precedenti si è provato a definire un modello di Data Mining che possa classificare situazioni particolari di tempo, di relativa importanza, per ottenere delle previsioni di alta accuratezza, riguardanti situazioni future, che si possono verificare avendo una data situazione pregressa già nota. Per effettuare la classificazione è necessario disporre di un dataset abbastanza grande, il quale può essere trasformato ottenendo una versione contente la concezione di successioni di eventi basati sul tempo, chiamate sequenze temporali (come definito nel paragrafo 3.3).

L'implementazione del classificatore, come descritto nel paragrafo 3.6, può essere fatta in diversi modi e con diverse tipologie di classificatore, in modo da provare a migliorarne le prestazioni.

Il primo classificatore impiegato nello sviluppo è quello definito all'interno della classe Classifier (paragrafo 3.6.4), che gestisce due tipologie diverse di sequenze temporali. La prima tipologia è quella definita dalle stazioni aventi i seguenti stati: Full, Empty e Normal. La seconda tipologia serve per poter distinguere meglio quelle situazioni che sono quasi al limite, cioè di quelle stazioni che sono quasi nello stato Full e quasi nello stato Empty, attribuendo ad esse lo stato di AlmostFull o AlmostEmpty.

Un altro classificatore affrontato nel paragrafo 3.7, chiamato TimeslotClassifier, permette di suddividere le sequenze di training e di test in fasce temporali, per poter designare solo una porzione delle regole estratte su cui verrà effettuata la predizione. Questo tipo di classificatore permette di dare maggiore rilevanza a regole che sono state generate partendo da fascia oraria, così facendo, il dato di test che ricade in quella fascia oraria verrà classificato utilizzando solo le regole che sono appartenti all'intervallo di tempo in questione. Anche in questo caso, c'è la possibilità di utilizzare le due tipologie di sequenze temporali che sono state descritte.

Le problematiche rilevate nel paragrafo 4.1.5, sono derivate da record mancati e record che hanno valori dei campi used e free non coerenti tra di loro al variare del tempo. A questo seguono problemi di prestazioni, poiché viene influenzata l'assegnazione corretta dello stato di una stazione durante la manipolazione del dataset e durante la predizione.

Nel paragrafo 4.3 è possibile, visionare le prestazioni del classificatore descritto all'intero della classe Classifier e TimeslotClassifier sulle prime dieci stazioni, andando a considerare le due tipologie di sequenze temporali. I risultati di questi esperimenti portano alla conclusione che vi è qualcosa di anomalo in queste prime dieci stazioni, poiché i risultati cambiano di poco tra loro al variare dei parametri impiegati per la generazione del dataset, come ad esempio il valore di granularità per assegnare la grandezza di una finestra temporale in cui le stazioni ricadono e il valore di soglia relativo all'assegnazione degli stati di AlmostFull e AlmostEmpty.

I risultati sperati sono all'aumentare del valore di granularità, le prestazioni del classificatore in gioco decrescano. Invece, per il caso del parametro di soglia numBikes_Th ci aspettiamo che le prestazioni della classificazione aumentino e poi decrescano arrivati ad un certo punto.

Nel paragrafo 4.4 si affronta la problematica di quale sia la migliore configurazione che si può attribuire alle classi Classifier e TimeslotClassifier. Dai risultati relativi a tutte le stazioni in gioco, si può trarre la conclusione che il miglior parametro di granularità è pari a 5 minuti. Alla classe Classifier si può attribuire come valore di minimo supporto per l'estrazione di regole di associazione 0.0001, che è il valore che porta risultati migliori. Tuttavia, nella classe TimeslotClassifier, il miglior valore di minimo supporto è pari a 0.001, poiché la suddivisione in quattro fasce temporali non porta a soddisfare il vincolo imposto sul numero di regole estratte con un minimo valore di supporto pari a 0.0001.

Per la generazione degli stati di AlmostFull e AlmostEmpty, il migliore valore di soglia descritto da numBikes_Th analizzato è pari a 4.

Nel grid search ci conferma, che all'aumentare del valore di granularità, le prestazioni della classificazione diminuiscono, anche se di poco.

Nella classe TimeslotClassifier, utilizzando un set di fasce orarie più stringenti, si ottiene un migliormanto delle prestazioni, confermando quanto detto in precedenza, e cioè che in alcuni casi per la classificazione di un dato di test, le regole di associazione che ricadono nelle fasce orarie diverse non sono utili nella classificazione.

Un altro problema relativo alla qualità del dataset utilizzato è lo sbilanciamento notevole delle classi che viene generato durante la trasformazione, in quasi tutti i casi dei valori bassi di richiamo per lo stato Full e Empty.

## 5.2 Implementazioni future

Un possibile sviluppo futuro potrebbe essere quello di provare ad utilizzare un'altra tipologia di dataset con una dimensione maggiore, per poter definire, in aggiunta, un validation set per compiere la validazione degli hyperparameters. Un altro possibile sviluppo potrebbe essere trovare una modalità di ripiego per gestire i salti temporali che vi sono all'interno del dataset e gestire i valori di used e free di una relativa stazione non coerenti al variare del tempo. Un'implementazione futura riguardante la classificazione può essere quella di introdurre il model ensemble, che permette di avere più classificatori in parallelo, attribuendo ad ognuno di essi un set di regole

di associazione, create su fasce orarie diverse. La predizione finale può essere data da un voto di maggioranza o da un voto pesato, in base alla fascia oraria su cui si vuole predire lo stato della stazione.

# Bibliografia


[1] Jiawei Han et al. *FreeSpan*. URL: https://www.cs.sfu.ca/~jpei/publications/freespan.pdf.

[2] Elena Baralis Andrea Pasini. *Introduction to Python*. URL: http://dbdmg.polito.it/dbdmg_web/wp-content/uploads/2019/10/DSL-Python-Introduction.pdf.

[3] *Apache Spark*. URL: https://spark.apache.org/.

[4] Elena Baralis. *Classification fundamentals*. URL: https://dbdmg.polito.it/wordpress/wp-content/uploads/2018/10/8-DMClassification.pdf.

[5] Elena Baralis. *Data science, The Big Data challenge*. URL: https://dbdmg.polito.it/wordpress/wp-content/uploads/2018/10/1-DSIntro.pdf.

[6] Elena Baralis. *The data mining process*. URL: https://dbdmg.polito.it/wordpress/wp-content/uploads/2018/10/5-DMProcess.pdf.

[7] Silvia Chiusano Elena Baralis. *Association Rules Fundamentals*. URL: https://dbdmg.polito.it/wordpress/wp-content/uploads/2019/10/DSL-3-MassRules.pdf.

[8] *FP-Growth*. URL: https://spark.apache.org/docs/latest/ml-frequent-pattern-mining.html.

[9] Paolo Garza. *Introduction to Spark*. URL: https://dbdmg.polito.it/wordpress/wp-content/uploads/2020/03/10_SparkIntroduction_BigData_2x.pdf.

[10] Paolo Garza. *RDD-based programming*. URL: https://dbdmg.polito.it/wordpress/wp-content/uploads/2020/03/11_SparkRDD_Basic_BigData_2x.pdf.

[11] Paolo Garza. *RDDs and key-value pairs*. URL: https://dbdmg.polito.it/wordpress/wp-content/uploads/2020/04/12_SparkRDD_PairRDD_BigData_2x.pdf.

[12] Paolo Garza. *Spark SQL*. URL: https://dbdmg.polito.it/wordpress/wp-content/uploads/2020/04/15_SparkSQL_Datasets_BigData_2x.pdf.

[13] *Matplotlib*. URL: https://matplotlib.org/.

[14] *Python*. URL: https://www.python.org/.

[15] *Scikit-learn*. URL: https://scikit-learn.org/stable/.

[16] Mohanmmed J. Zaki. *SPADE*. URL: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=B4EDB00CBF677F73C37AD1DEC73E7310?doi=10.1.1.113.6042&rep=rep1&type=pdf.