ANALISI DELLE TRANSAZIONI

Componenti del gruppo

- Lattaruli Armando, 762917, a.lattaruli2@studenti.uniba.it
- Giustino Alessio, 758353, a.giustino11@studenti.uniba.it

Link GitHub: progetto

A.A. 2023-2024

Indice

Capitolo 0) Introduzione

L'obiettivo di questo progetto è quello di riorganizzare un insieme di transazioni avvenute tra il 01/12/2010 e il 09/12/2011 in classi naturali mediante apprendimento non supervisionato e successivamente addestrare un modello che sia in grado di inserire una nuova transazione nella propria classe naturale mediante apprendimento supervisionato.

Requisiti funzionali

Il progetto è stato realizzato in Python, in quanto è un linguaggio che mette a disposizione molte librerie che permetto di trattare i dati in modo facile e intuitivo.

- Versione Python: 3.11
- IDE utilizzato: VSCode
- Librerie utilizzate:
  - **matplotlib**: visualizzazione dei grafici (curve di apprendimento, grafici a barre, grafici a torta)
  - **numpy**: libreria per gestire array
  - **pandas**: importazione dei dataset in formato *.csv*
  - **scikit_learn**: libreria utilizzata per apprendimento automatico
  - **kneed**: libreria utilizzata per identificare il gomito di una funzione

Installazione e avvio

Aprire il progetto con l'IDE preferito. Avviare il programma partendo dal file main.py

Capitolo 1) Creazione del dataset

Il dataset utilizzato è stato trovato su Kaggle.

Per quanto riguarda le features utilizzate in questo progetto abbiamo deciso di utilizzare le stesse che Spotify usa per descrivere le sue canzoni nella documentazione:

- **InvoiceNo** → Numero di fattura
- **StockCode** → Codice di magazzino
- **Description** → Descrizione dell'item
- **Quantity** → Quantità acquistata
- **InvoiceDate** → Data della fattura
- **UnitPrice** → Prezzo della singola unità
- **CustomerID** → Codice identificativo del cliente
- **Country** → Paese in cui è avvenuta la transazione

Preprocessing del dataset

Il preprocessing del dataset è stato utilizzato per la creazione del dataset realmente utilizzato successivamente. Tutte le feature che sono descritte mediante delle stringhe alfanumeriche sono state convertite in un formato intero mediante il metodo **factorize** messo a disposizione dalla libreria *pandas*. Sono state eliminate successivamente tutte le righe con almeno un valore nullo all'interno. Nella seconda parte dell'esperimento sono state eliminate dal dataset le feature **InvoiceNo, InvoiceDate, CustomerID** in quanto, secondo noi, non sono feature che servono per suddividere al meglio le transazioni.

Capitolo 2) Apprendimento non supervisionato

L'apprendimento non supervisionato è una branca dell'apprendimento automatico in cui l'agente viene addestrato su un insieme di dati *senza etichette*. Esistono due principali tipi di apprendimento non supervisionato:

- **Clustering** → L'obiettivo è raggruppare gli elementi del dataset in base a delle somiglianze
- **Riduzione della dimensionalità** → L'obiettivo è ridurre il numero di feature utilizzate mantenendo però le informazioni più significative

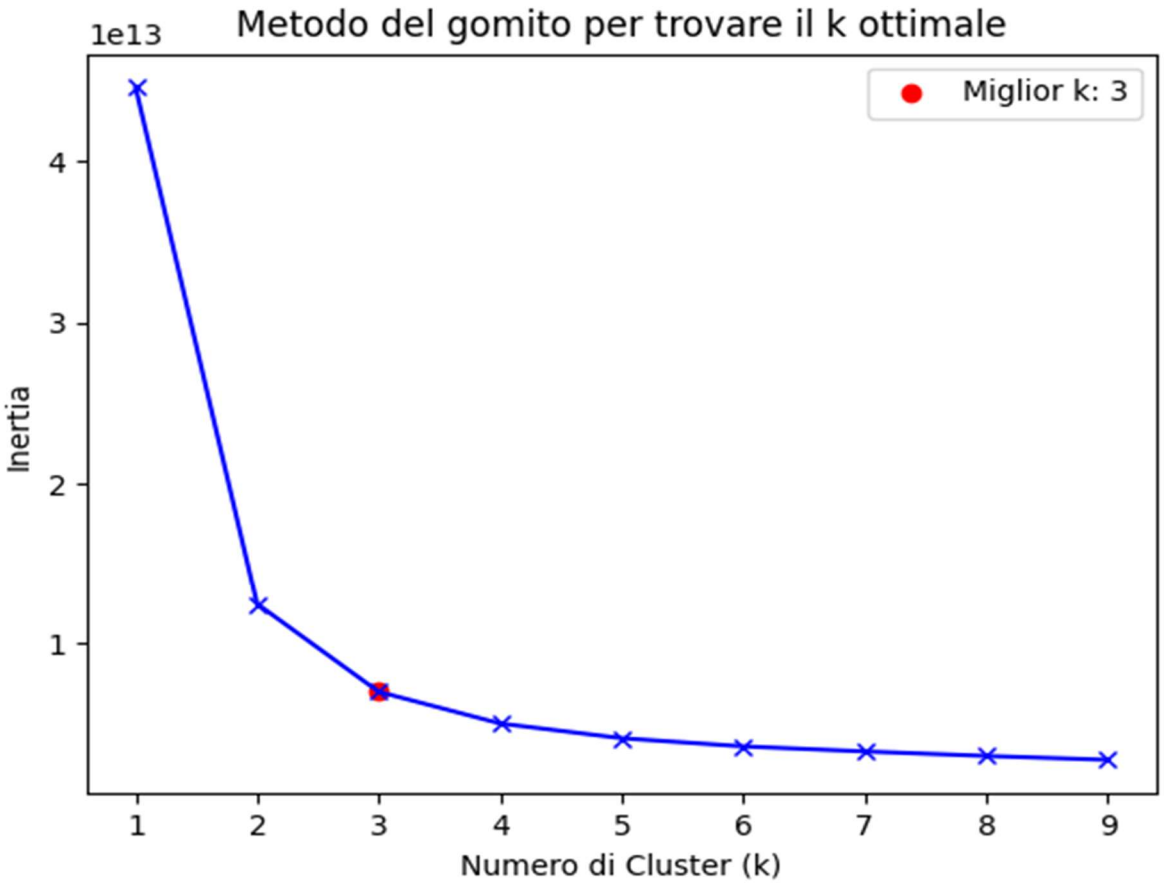
Nel nostro caso l'apprendimento non supervisionato è stato utilizzato con lo scopo di effettuare clustering, per raggruppare le transazioni in dei gruppi che formeranno le classi naturali. Esistono due tipologie di clustering: **hard clustering** e **soft clustering**. Il primo associa ogni esempio a un cluster specifico, il secondo associa a ogni esempio la probabilità di appartenenza a ogni cluster.

Per la realizzazione di questa parte è stata usata la libreria *sklearn* per il clustering, la libreria *matplotlib* per la visualizzazione di grafici e la libreria *kneed* per individuare il gomito.

Nel nostro caso abbiamo deciso di utilizzare l'algoritmo di hard clustering *KMeans*. Uno dei problemi principali del *KMeans* è trovare il numero ideale di cluster da fornire in input all'algoritmo. Per risolvere questo problema abbiamo seguito una strategia nota come "curva del gomito". La curva del gomito mostra la variazione dell'*inertia* al variare del numero di cluster, dove l'*inertia* rappresenta la somma delle distanze quadrate tra ogni punto dati e il centro del cluster assegnato. L'algoritmo di prevede dunque di eseguire il *KMeans* per diversi valori di numero di cluster, calcolare l'*inertia* per ognuno di essi, plottare su un grafico la curva e poi identificare il gomito, ovvero il punto in cui la diminuzione dell'*inertia* diventa meno significativa.

```python
def regolaGomito(dataSet):
    intertia = []
    maxK=10
    for i in range(1, maxK):
        kmeans = KMeans(n_clusters=i,n_init=5,init='random')
        kmeans.fit(dataSet)
        intertia.append(kmeans.inertia_)
```

Per quanto riguarda le scelte progettuali abbiamo deciso di eseguire l'algoritmo con 10 valori di cluster, che variano da 1 a 10. Per quanto riguarda il parametro *n_init,* questo indica il numero delle "RANDOM_RESTART". In particolar modo per ogni numero di cluster **i** verrà eseguito l'algoritmo per 5 volte e verrà restituito il clustering migliore. Per quanto riguarda il parametro *init='random'* significa che i centroidi vengono inizializzati in maniera del tutto casuale.



Nel nostro caso possiamo vedere come il numero ottimale di cluster è *k = 3*, dunque successivamente è stato eseguito l'algoritmo *KMeans* con 3 cluster.

Ogni cluster viene identificato da un indice.

Capitolo 3) Apprendimento supervisionato

L'apprendimento supervisionato è una branca dell'apprendimento automatico in cui l'agente viene addestrato su un insieme di dati con etichette. Si divide in

- **Classificazione**: Le etichette rappresentano un dominio di valori discreto (quindi un insieme finito di valori) che la feature di output può assumere. Un caso particolare è la classificazione booleana, in cui le classi sono **true** e **false**.
- **Regressione**: L'etichetta può essere un qualsiasi valore numerico, quindi il dominio della feature di output è continuo.

L'obiettivo principale è far sì che l'algoritmo impari una relazione tra gli input e gli output in modo che, una volta addestrato, possa fare previsioni accurate su nuovi dati mai visti prima.

Ci sono diverse fasi da affrontare:

1) Scelta degli iper-parametri
2) Fase di addestramento
3) Fase di test
4) Valutazione delle prestazioni

Per questa parte di progetto sono state utilizzate le seguenti librerie: *matplotlib* e *numpy* per la visualizzazione di grafici, *sklearn* per l'apprendimento supervisionato.

Nel nostro caso l'apprendimento supervisionato è stato utilizzato con scopo di classificazione, dove la classi sono gli indici dei cluster prima ottenuti mediante apprendimento non supervisionato.

Per questo progetto abbiamo deciso di utilizzare **tre modelli** di apprendimento automatico:

- **DecisionTree** → Classificatore strutturato ad albero in cui le foglie rappresentano le classi di appartenenza (o le probabilità di appartenenza a tali classi) mentre la radice e i nodi interni rappresentano delle condizioni sulle feature di input. A seconda se tali condizioni sono rispettate o meno, verrà seguito un percorso piuttosto che un altro e alla fine si arriverà alla classe di appartenenza
- **RandomForest** → Classificatore che si ottiene creando tanti DecisionTree. Il valore in output si ottiene mediando sulle predizioni di ogni albero appartenente alla foresta (tecnica di bagging)

- **Regressione Logistica Multinomiale (Funzione lineare)** → A differenza della classica regressione lineare, che è utilizzata per predire valori continui (regressione), la regressione logistica multinomiale viene utilizzata per calcolare le probabilità di appartenenza di un esempio a ogni classe. Per ogni classe abbiamo una funzione lineare di cui apprendere i pesi mediante tecnica di discesa di gradiente. In ogni funzione vi è un peso per ogni feature più un termine noto w_0 (a cui associamo una feature immaginaria X_0 con valore costante 1). Il risultato viene "schiacciato" da una funzione nota come **softmax**, la quale trasforma un vettore di valori reali in una distribuzione di probabilità su più classi. Il risultato è la classe con la probabilità maggiore

Fase 1) Scelta degli iper-parametri

Gli iper-parametri sono i parametri di un modello di apprendimento automatico, i quali non vengono appresi durante la fase di addestramento come i normali parametri del modello (es. i pesi di una funzione lineare) ma devono essere necessariamente fissati prima che il modello possa cominciare l'addestramento. La loro scelta influisce sulle prestazioni e sulla complessità del modello. Uno dei compiti più complessi è proprio la scelta degli iper-parametri per i vari modelli.

Per la scelta degli iper-parametri abbiamo utilizzato una tecnica di *K-Fold Cross Validation* (CV).

Nella *K-Fold CV* il dataset viene diviso in *k* fold (insiemi disgiunti) e il modello viene addestrato *k* volte. Per ogni iterazione 1 fold viene usato per il testing mentre gli altri *k-1* fold vengono utilizzati per il training. In questo modo è possibile testare e addestrare il modello su dati diversi per comprendere "la bontà" del modello.

La strategia che abbiamo deciso di applicare per ricercare gli iper-parametri dei nostri modelli è la *GridSearch* con *Cross Validation*. In questo approccio vengono definite le griglie dei valori possibili per gli iper-parametri e si esplorano tutte le combinazioni possibili alla ricerca della miglior combinazione possibile.

IPER-PARAMETRI RICERCATI e UTILIZZATI

DecisionTree

- *Criterion* → Misura la qualità dello spit effettuato sui nodi. Può assumere i seguenti valori:
    - gini → Misura la "purezza" della divisione dei dati, più precisamente quanto spesso un elemento viene classificato in modo sbagliato

    - entropy → Misura la quantità di disordine nei dati. Minimizzare l'entropia significa massimizzare l'informazione guadagnata durante ***che cosa?***
    - log_loss → perdita logaritmica. Indicata quando l'output corrisponde a una probabilità piuttosto che a un valore di classe.
- *Splitter* → Indica la strategia da utilizzare per il criterio di split. In questo caso abbiamo usato il valore di default **best** che indica il miglior criterio di split possibile, e dunque non l'abbiamo ricercato ma l'abbiamo direttamente utilizzato.
- *Max_depth* → Indica l'altezza massima dell'albero
- *Min_samples_split* → Il numero minimo di esempi necessari affinché possa essere inserito un criterio di split. Se il numero è minore viene innestata una foglia
- *min_samples_leaf* → Il numero minimo di esempi per poter creare una foglia

```
DecisionTreeHyperparameters = {
    'criterion': ['gini', 'entropy','log_loss'],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5, 10],
    'splitter': ['best']}
```

RandomForest

Per quanto riguarda la creazione dei singoli alberi abbiamo utilizzato gli stessi criteri del DecisionTree (meno splitter che non è un iperparametro per questo modello). Per quanto riguarda i criteri relativi alla foresta abbiamo:

- *n_estimators*: il numero di alberi nella foresta

```
RandomForestHyperparameters = {
    'criterion': ['gini', 'entropy','log_loss'],
    'n_estimators': [2, 4,6],
    'max_depth': [5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5]}
```

LogisticRegression (Multinominal)

- *C* → Specifica quanto "forte" è la regolarizzazione. Più il valore di C è basso e più forte è la regolarizzazione. Valori di C piccoli prevengono *overfitting* ma rendono il modello più rigido. Valori alti rendono il modello più flessibile ma sono soggetti a *overfitting*

- *Max_iter* → Indica il numero di iterazioni utilizzate
- *Solver* → Indica l'algoritmo da utilizzare per apprendere i pesi
    - lbfgs → mantiene solo alcune informazioni delle derivate parziali, utilizzando dunque poca memoria. È adatto per problemi con numero di feature elevato, soprattutto quando la memoria a disposizione non è molta
    - liblinear → Basato su algoritmi di programmazione lineare

```
LogisticRegressionHyperparameters = {
    'C': [0.01, 0.1, 1],
    'solver': ['liblinear', 'lbfgs'],
    'max_iter': [100,200]}
```

Nel nostro caso ogni modello è stato addestrato con un valore di *K = 3 (3 fold, 3 iterazioni)*

I parametri che sono stati restituiti sono:

Modello	Parametro	Valore
DecisionTree	criterion	gini
	max_depth	10
	min_samples_leaf	1
	min_samples_split	2
	splitter	best
RandomForest	criterion	log_loss
	max_depth	10
	min_samples_leaf	1
	min_samples_split	10
	n_estimators	6
LogisticRegression	C	0.1
	max_iter	100
	solver	liblinear

Fasi 2) e 3) Fase di addestramento e test

Per la fase di addestramento e di test i modelli sono stati addestrati utilizzando una *K-Fold Cross Validation* con *K = 3*

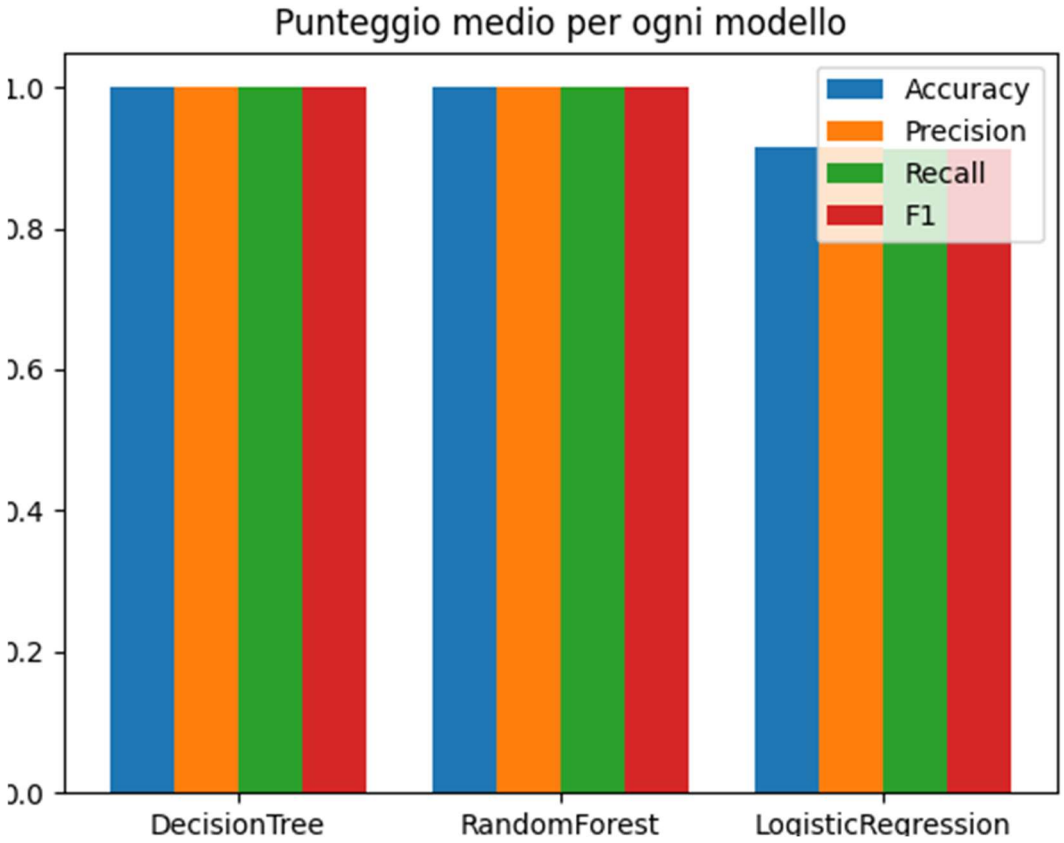
Fase 4) Valutazione delle prestazioni

La K-Fold Cross Validation, come già detto precedentemente, permette di addestrare il modello diverse volte utilizzando ogni volta esempi di training e di test diversi, in modo tale da ottenere delle valutazione più "veritiere" rispetto alle reali performance del sistema.

Per la valutazione delle performance del sistema abbiamo deciso di utilizzare le seguenti metriche:

- ***Accuracy*** → L'accuracy è una misura generale della correttezza del modello e rappresenta la frazione di previsioni corrette rispetto al totale delle previsioni.
- ***Precision_Macro*** → La precision macro è la media delle precisioni calcolate per ogni classe. La precisione per ogni classe è il numero di istanze di classe **c** classificate nella classe **c** / numero di istanze classificate nella classe **c**
- ***Recall_Macro*** → La recall macro è la media delle recall calcolate per ogni classe. La recall per ogni classe è il numero di istanze di classe **c** classificate nella classe **c** / numero di istanze nella classe **c**
- ***F1_Macro*** → La F1 macro è la media delle F1 calcolate per ogni classe. La F1 calcolata per ogni classe è la media armonica tra precision e recall (dunque è alta solo se entrambi i valori sono alti) ed è calcolata come segue: 2*Precision*Recall/(Precision+Recall)

RISULTATI



Possiamo notare che in generale i valori sono molto buoni per ogni metrica in ogni modello. Questo potrebbe rappresentare un segnale di *overfitting*. Per verificare la presenza di *overfitting* bisogna prendere in considerazione altri fattori, come la varianza, la deviazione standard e le curve di apprendimento.

L'*overfitting* è un problema molto noto in apprendimento automatico. Si verifica quando un modello si sovra-adatta ai dati di addestramento, imparando correlazioni spurie presenti nel dataset e non riuscendo a generalizzare bene (e dunque il modello non sarà in grado di predire correttamente i valori per esempi mai visti che non presentano queste correlazioni).

Varianza e deviazione standard delle curve di apprendimento

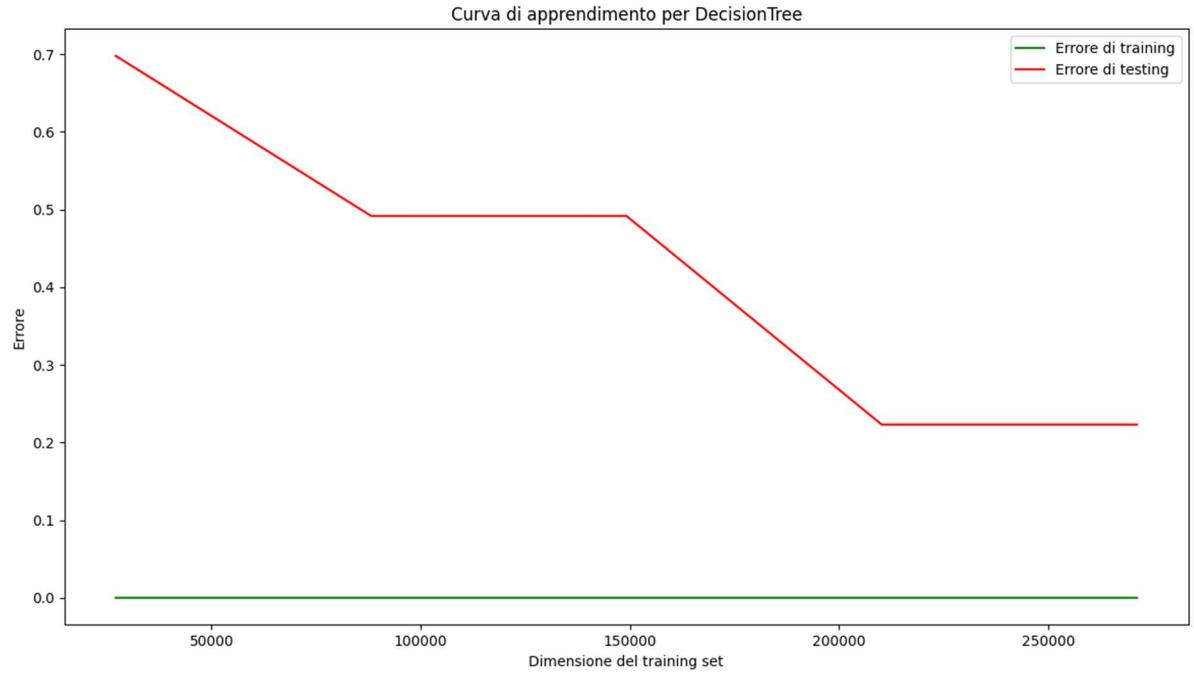
Varianza → È la media dei quadrati delle differenze tra ciascun dato e la media

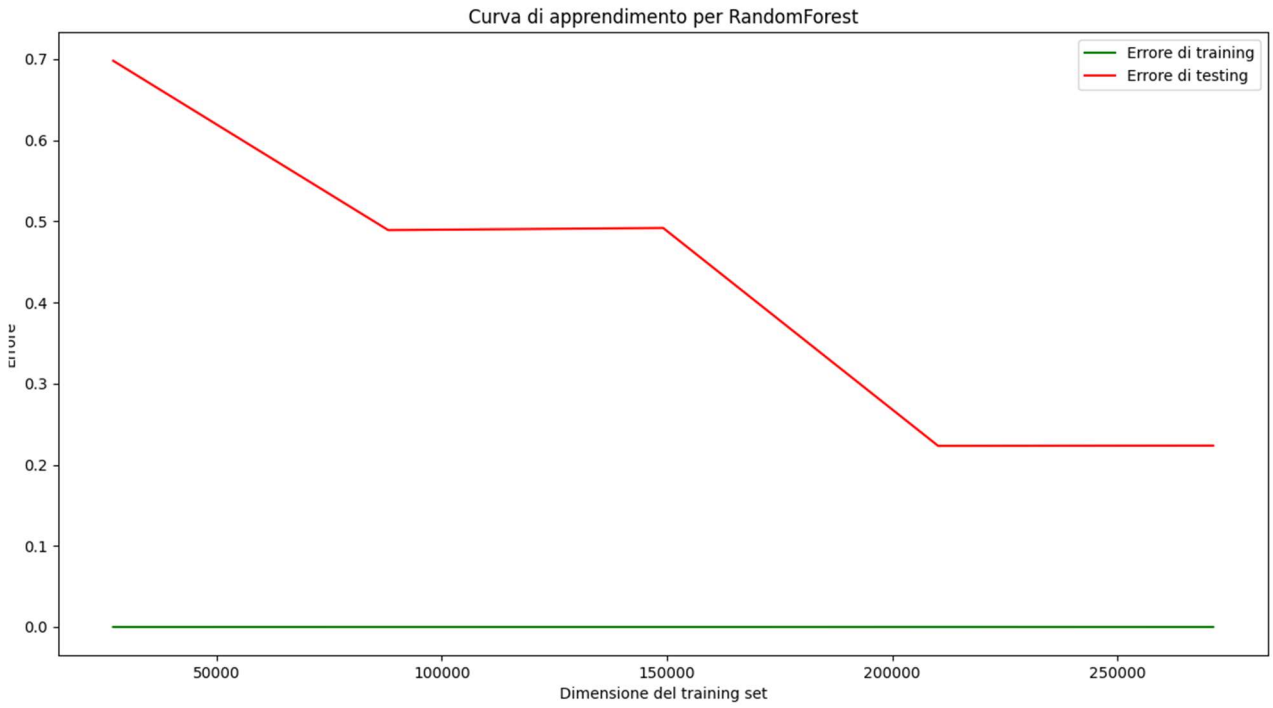
Deviazione Standard → Radice quadrata della varianza

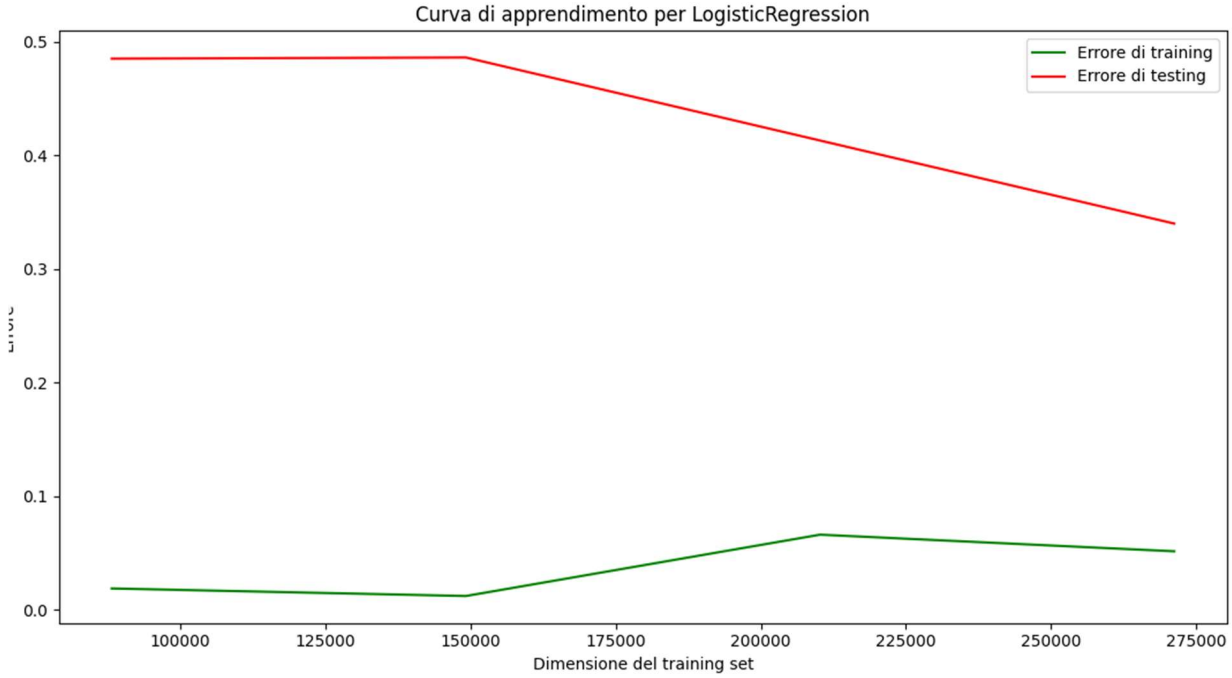
Nel contesto dell'apprendimento automatico misurano entrambe la dispersione dell'errore. Alti valori suggeriscono che il modello è sensibile alle variazioni dei dati in input, e dunque suggeriscono una possibile presenza di *overfitting*.

Modello	Tipo di Errore	Errore Std	Varianza Errore
DecisionTree	Train Error	1.738095e-06	3.020976e-12
	Test Error	0.158768	0.025207
RandomForest	Train Error	3.650001e-05	1.332250e-09
	Test Error	0.159047	0.025296
LogisticRegression	Train Error	0.049515	0.002452
	Test Error	0.227290	0.051661

Curve di apprendimento







Analisi dei risultati per ogni modello

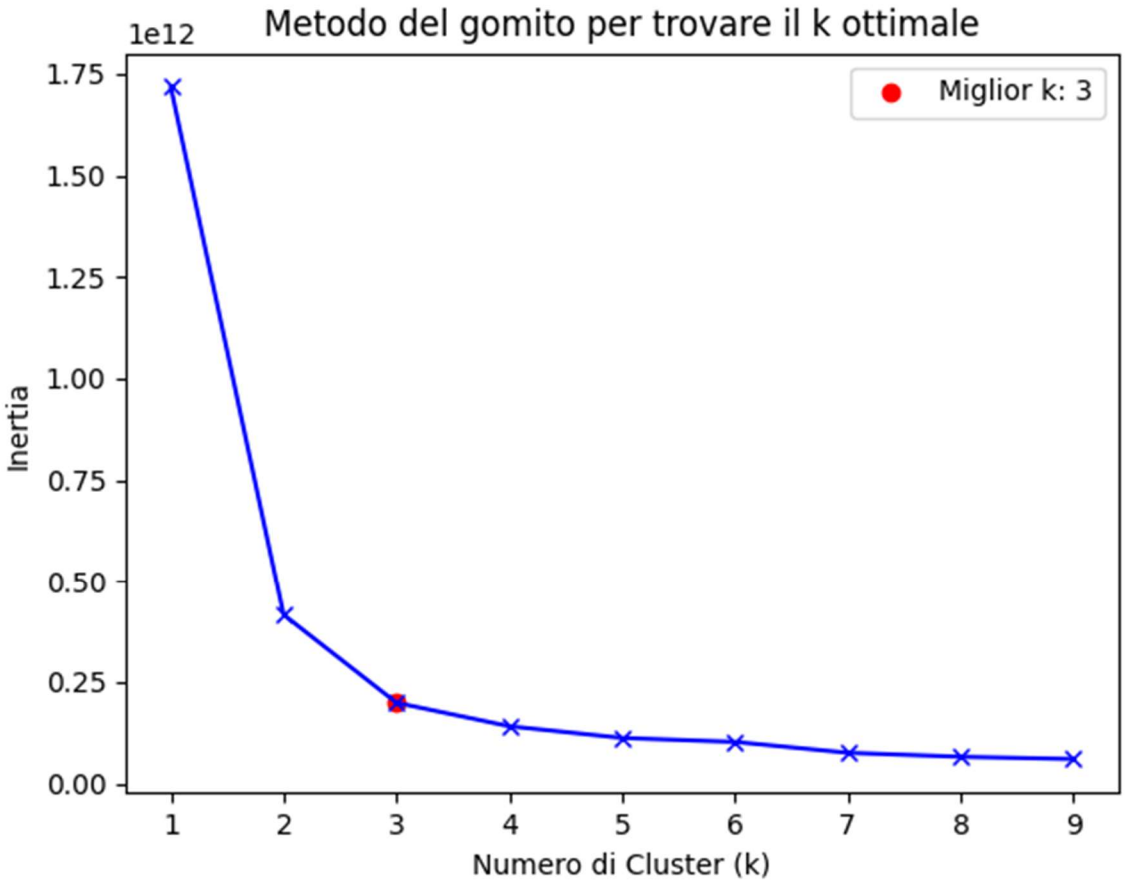
- **Decision Tree:** I valori di deviazione standard e varianza sono relativamente basse per la curva di training mentre sono molto più alte per la curva di test. Osservando anche le curve si può concludere che ci troviamo davanti a un caso di underfitting (cioè il modello non riesce a generalizzare bene perché ci sono pochi dati di buona qualità a disposizione da cui imparare)

- **Random Forest:** La situazione è analoga a quella del Decision Tree.
- **Logistic Regression:** In questo caso anche i valori per la curva di training sono elevati

Al momento il miglior classificatore sembrerebbe essere il DecisionTree in quanto presenta gli errori minori.

POSSIBILE SOLUZIONE

Come già accennato per provare a migliorare i risultati si è deciso in modo arbitrario di eliminare le feature che, logicamente parlando, non dovrebbero essere considerate per divedere le transazioni in classi naturali. Queste feature sono: **InvoiceNo, InvoiceDate, CustomerID**



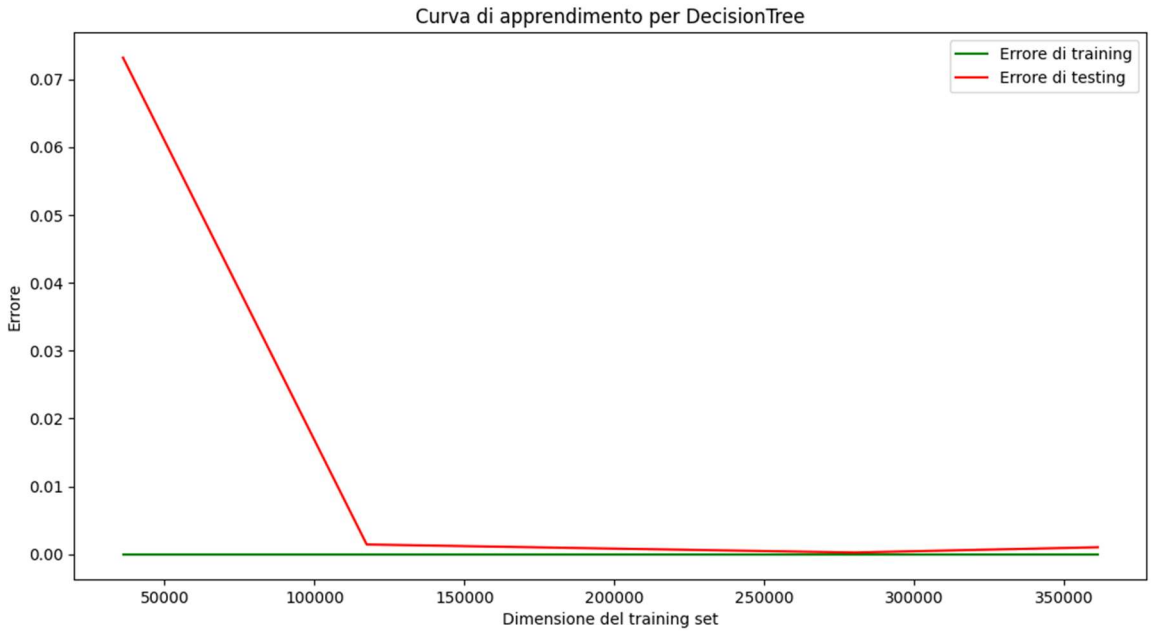
Anche in questo caso sono state individuate 3 classi naturali. Possiamo vedere come l'inertia risulta differente rispetto allo scorso esperimento, dunque eliminare 3 feature ha cominciato a dare dei risultati.

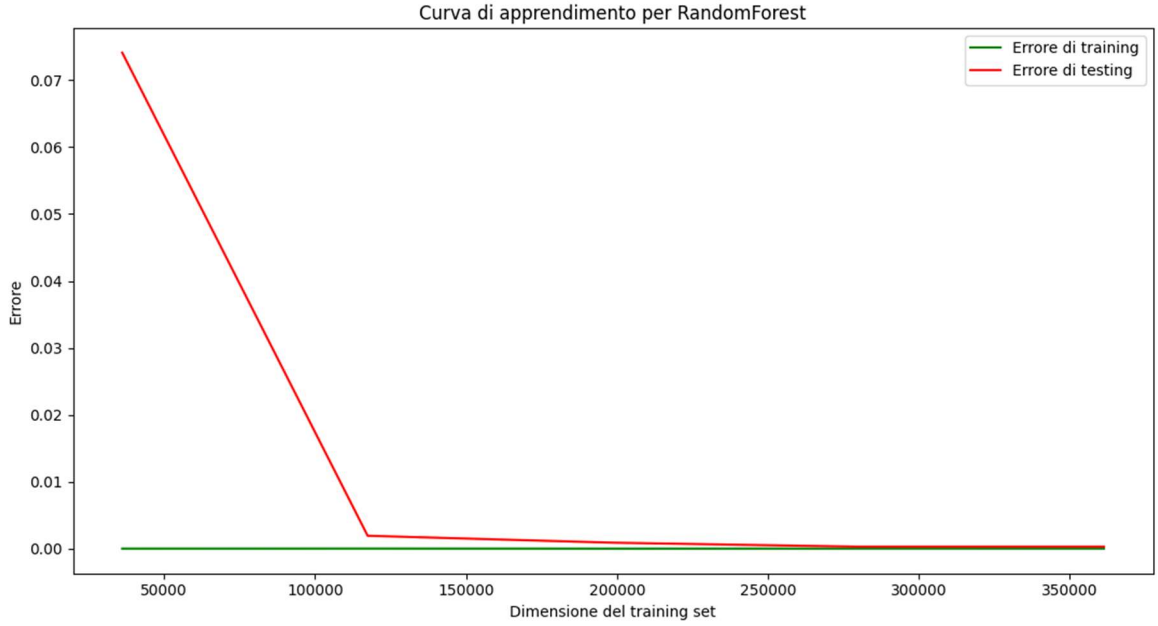
Modello	Parametro	Valore
DecisionTree	criterion	entropy
	max_depth	10
	min_samples_leaf	1
	min_samples_split	2
	splitter	best
RandomForest	criterion	entropy
	max_depth	10
	min_samples_leaf	1
	min_samples_split	10
	n_estimators	6
LogisticRegression	C	1
	max_iter	200
	solver	liblinear

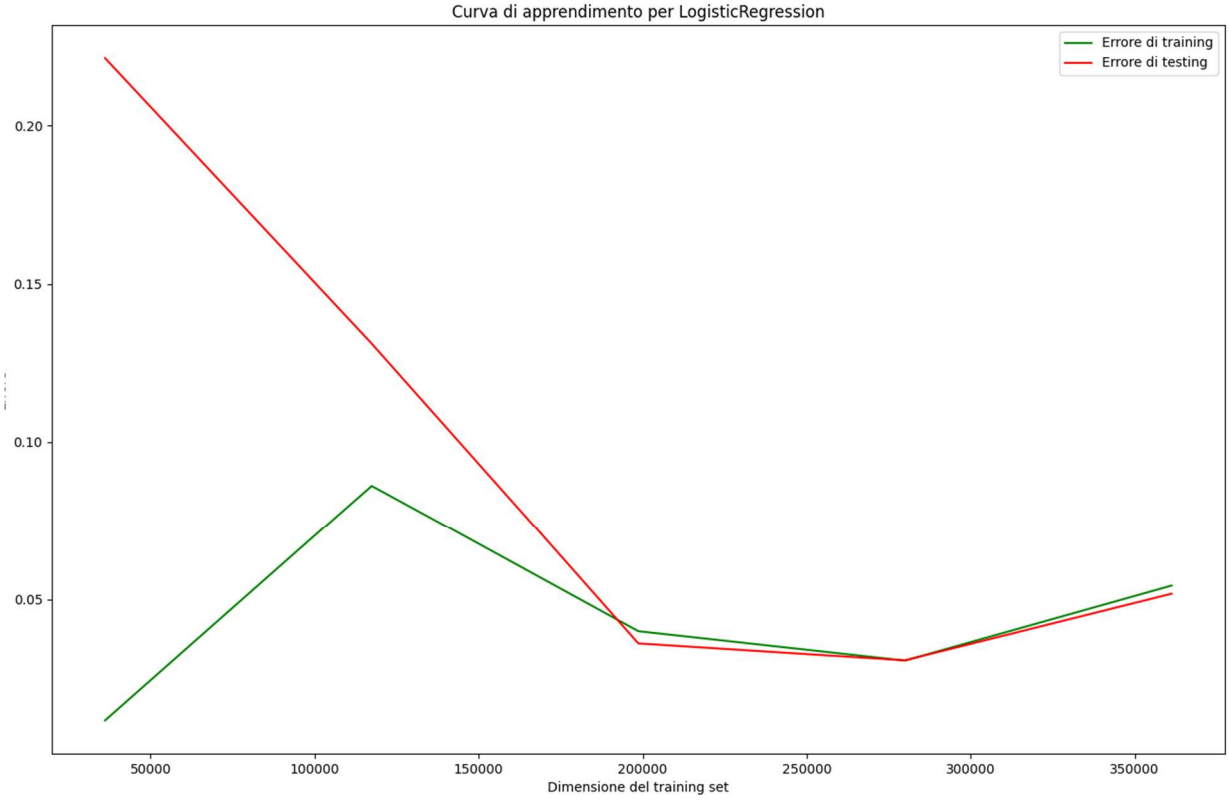
La GridSearchCV ha restituito dei valori diversi rispetto al dataset precedente

Modello	Tipo di Errore	Errore Std	Varianza Errore
DecisionTree	Train Error	0.0	0.0
	Test Error	0.001300	1.691204e-06
RandomForest	Train Error	2.050727e-05	4.205483e-10
	Test Error	0.000171	2.928489e-08
LogisticRegression	Train Error	0.009576	9.169553e-05
	Test Error	0.012612	1.590655e-04

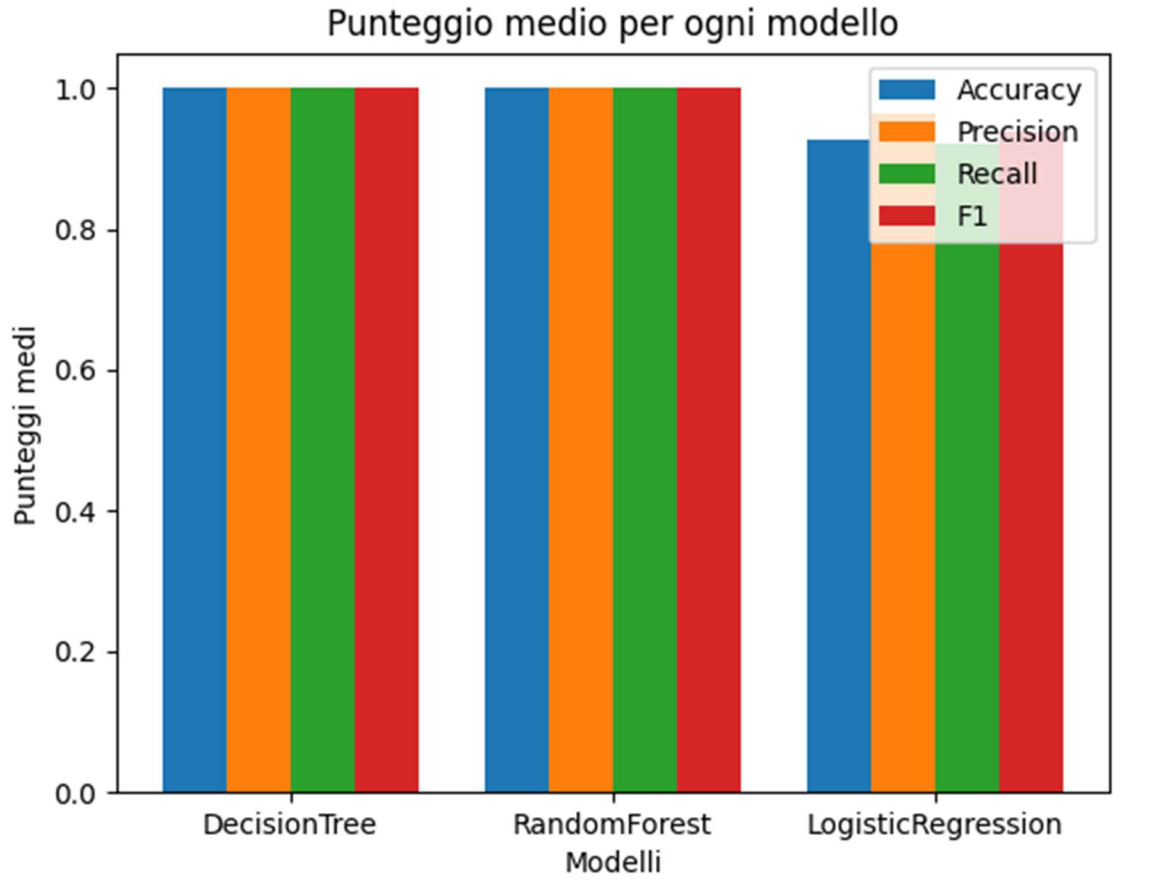
Gli errori sono diminuiti rispetto all’esperimento precedente


Curva di apprendimento per DecisionTree
Errore di training
Errore di testing
Errore
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
50000
100000
150000
200000
250000
300000
350000
Dimensione del training set


Curva di apprendimento per RandomForest
Errore di training
Errore di testing
Errore
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
50000
100000
150000
200000
250000
300000
350000
Dimensione del training set



Le curve sono migliorate di molto.



Conclusione: L'eliminazione delle feature sopra elencate ha portato a notevoli cambiamenti in positivo per il nostro esperimento. Osservando gli errori, le curve e i punteggi delle metriche possiamo concludere che il ***DecisionTree*** (con il dataset ridotto) rappresenta il modello migliore.

Sviluppi futuri

Si potrebbe applicare la *PCA* o altre tecniche di *feature selection* per provare a determinare in modo matematico quali feature eliminare per migliorare il modello. Si potrebbe anche costruire una rete bayesiana e confrontarla con il miglior risultato ottenuto dai modelli di apprendimento supervisionato.

Riferimenti bibliografici

- Apprendimento supervisionato: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. Cambridge University Press. 3rd Ed. [Ch.7]
- Apprendimento non supervisionato: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. Cambridge University Press. 3rd Ed. [Ch.10]
- Dataset: https://www.kaggle.com/datasets/hellbuoy/online-retail-customer-clustering