Companion Finanziario e Predittore di Frodi

Gruppo di lavoro:
Zippo Dario, 747006, d.zippo4@studenti.uniba.it

Repository progetto:

DarioZippo01/Financial-Companion

Documentazione progettuale

AA 2022-23

Sommario

Capitolo 1: Introduzione 3
Capitolo 1.1: Requisiti funzionali 3
Capitolo 1.2: Argomenti di interesse 4
Capitolo 2: Creazione del Dataset 6
Capitolo 2.1: Preprocessing del Dataset: 7
Capitolo 3: Apprendimento Supervisionato 10
Capitolo 3.1: Ottimizzazione degli Iper-Parametri 11
Capitolo 3.2: Addestramento e testing del Modello 16
Capitolo 3.3: Valutazione delle prestazioni 16
Capitolo 3.4: Analisi dei risultati: 25
Capitolo 4: Inclusione del Large Language Model 28
Capitolo 4.1: Introduzione ai Large Language Models 29
Capitolo 4.2: Feed Forward Neural Network nei Transformers 29
Capitolo 4.3: Implementazione del Large Language Model 31
Capitolo 4.4: Risultati 33
Capitolo 5: Apprendimento non Supervisionato 34
Capitolo 5.1: Motivazioni, insieme di input e insieme di output: 35
Capitolo 5.2: Ottimizzazione dei clusters: 36
Grafico dei BIC per ogni k cluster 37
Capitolo 5.3: Applicazione del modello Gaussian Mixture Model e Risultati: 37
Capitolo 6: Ragionamento logico tramite Prolog 39
Capitolo 6.1: Introduzione al Ragionamento Logico 39
Capitolo 6.2: Implementazione della Knowledge Base 40
Sviluppi Futuri 42
Riferimenti bibliografici: 43

Capitolo 1: Introduzione

Il progetto nasce dalla volontà di creare un sistema che agisca da compagno finanziario per banche e aziende.
La veridicità delle transazioni che fanno affidamento ad un sistema informatico può essere compromessa dalla presenza di frodi.
Tali frodi possono, se permesse o non notate, progredire ad uno stato in cui producono un profitto per l'utente origine o destinatario e potrebbero essere anche irrecuperabili nei casi in cui la liquidità viene estratta dal conto in contanti.

Per questo motivo, questo sistema ha il compito di predire, quando una transazione viene effettuata, se essa sia o meno fraudolenta. Se fraudolenta, verrà fermata.
Se ambigua, invece, tale transazione verrà prima controllata da Large Language Model e, infine, processata o rifiutata se determinata come fraudolente.

Il compagno offre inoltre servizi di organizzazione, interrogazione e analisi delle transazioni, degli utenti che le svolgono e delle relazioni che possono essere tra di esse, tramite regole e query logiche su fatti in una Knowledge Base.

L'utente per cui è pensato il progetto è un'azienda finanziaria con dataset di transazioni catalogate per frodi o genuine.

Capitolo 1.1: Requisiti funzionali

Il sistema è scritto ne linguaggio di programmazione Python così da poter fare uso delle sue librerie di manipolazione di dataset e file .pl per l'uso di prolog, modelli supervisionati/non supervisionati e Large Language Models.
È stato implementato l'uso di file di tipo "Python Notebook" (.ipynb" per la manipolazione del dataset, così da poter avviare separatamente le azioni eseguite su di esso.

La versione utilizzata è Python 3.11 e il sistema è stato testato in IDE Visual Studio Code.

Librerie utilizzate

- **Pandas**: per l'importazione e la manipolazione dei dataset prelevati da **Kaggle**;
- **Math**: per la manipolazione dei numeri in fase di preprocessing;
- **Sickit-learn**: per l'inclusione dei modelli di apprendimento supervisionato e non supervisionato utilizzati;
- **Sickit-optimize**: per l'inclusione dell'algoritmo "Bayes Search CV" utilizzato per la ricerca degli iper-parametri;
- **Matplotlib**: per la creazione e visualizzazione dei grafici mostrati in documentazione;
- **Numpy**: per la gestione degli array e la produzione di valori come varianza e media derivanti dai valori in essi contenuti;
- **Openai**: per l'inclusione del LLM di OpenAI "GPT-3.5-Turbo";
- **Pyswip**: per l'esecuzione di query prolog da codice Python su Knowledge Base ".pl";
- **Jupyter Kernel**: Che non è una libreria ma un kernel che permette di avviare le celle dei file di tipo Python Notebook.

Installazione e avvio

Una volta scaricato e aperto il progetto, eseguire la fase di installazione dei requisiti – che installerà tutte le librerie necessarie sul vostro virtual environment – e poi tramite main.py avviare le diverse funzionalità del sistema. Il dataset base è assente poiché troppo pesante per la repository in Github quindi dovrà essere installato separatamente.

Per la gestione dell'apprendimento supervisionato, non supervisionato e preprocessing sono presenti, per ordine e precisione, codici .ipynb per la rapida esecuzione anche separata delle diverse fasi.

Capitolo 1.2: Argomenti di interesse

- **Creazione e manipolazione di un Dataset**:

È stato manipolato un dataset preesistente per poterlo adattare ai bisogni del sistema e quindi è stato trasformato in formato adatto all'input così da poter predire, inserita una transazione all'interno del sistema, se sia fraudolenta o meno.
Il dataset viene inoltre usato nelle fasi di apprendimento non supervisionato e nella conversione dataset.csv -> fatti.pl dei dati.

- **Apprendimento supervisionato**:

Sono stati istruiti 4 modelli di apprendimento supervisionato ed è stato selezionato il migliore di essi.
Il modello selezionato viene utilizzato per la classificazione dell'etichetta riguardante la natura genuina o fraudolenta di una transazione, sulla base del training set.

- **Apprendimento non supervisionato**:

Viene utilizzato, in un servizio differente dalla predizione e quindi in sede separata, l'apprendimento supervisionato per la creazione di cluster così da essere inviato come input alla fase di Programmazione Logica per l'analisi delle transazioni.

- **Programmazione Logica (Prolog)**:

Successivamente all'apprendimento non supervisionato, la programmazione logica è usata per la creazione di regole tra gli utenti e sé stessi, le transazioni e sé stesse, o tra utenti e transazioni.
Viene creata, partendo dal dataset preprocessato, una Knowledge Base per procedere in Prolog così da poter offrire servizi di Financial Companion all'utente.
Tali regole e i risultati dell'apprendimento non supervisionato vengono poi sfruttate per effettuare analisi sulle transazioni.

- **Introduzione alle Feed Forward Neural Network per le basi sui LLM/Transformers**:

Il sistema implementa un servizio basato sul modello Large Language Model, che applica apprendimento supervisionato e reti neurali feed forward per predire del testo. È utilizzato "OpenAI ChatGPT 3.5-Turbo" per l'analisi delle transazioni.
Nel caso in cui una transazione sia valutata come ambigua, allora un LLM avrà il compito di definire se essa sembri una frode o meno secondo gli standard la sua conoscenza.
Viene utilizzato un language model già definito, ergo non è trattata la fase di Deep Learning nelle neural networks all'interno del progetto ma viene introdotto per definire il funzionamento di un decoder nei transformers.

Capitolo 2: Creazione del Dataset

Dopo una ricerca riguardo i dataset disponibili sulla rete che contengano transazioni e le informazioni riguardo il loro stato di fraudolenza, è stato rilevato Synthetic Financial Datasets For Fraud Detection , un simulatore di transazioni in un lasso di tempo di 30 giorni, come input valido per il sistema.

Differentemente da quanto detto all'interno della documentazione del dataset, le colonne del bilancio vengono utilizzate poiché, nei casi reali, sono essenziali ai fini di rilevamento delle frodi.

Il dataset comprende transazioni partendo da uno step 1 ad uno step 743, e comprende quindi tutte le transazioni che accadono nel lasso di 743 ore e le risposte a tali transazioni, che sono a loro volta ulteriori transazioni.

Il dataset utilizzato contiene 6362621 elementi e presenta le seguenti colonne:

- **Step**: È un numero intero e descrive il momento, misurato in ore, all'interno di cui è stata effettuata la transazione;
- **Type**: È una stringa e definisce la tipologia di transazione effettuata. La transazione può avere ed è vincolata ad uno dei seguenti valori:
    - PAYMENT, CASH_OUT, CASH_IN, DEBIT, TRANSFER;
- **Amount**: È un numero reale e contiene l'ammonto - in dollari - di soldi interessati dalla transazione, indipendentemente dalla sua tipologia.
- **NameOrig**: È una stringa e descrive il nome dell'acconto da cui parte la transazione, che d'ora in poi chiameremo "mittente";
- **OldBalanceOrig**: È un numero reale e contiene il patrimonio dell'acconto mittente prima della transazione;
- **NewBalanceOrig**: È un numero reale e contiene il patrimonio dell'acconto mittente dopo la transazione;
- **NameOrig**: È una stringa e descrive il nome dell'acconto a cui il mittente indirizza la transazione, che d'ora in poi chiameremo "destinatario";
- **OldBalanceDest**: È un numero reale e contiene il patrimonio dell'acconto destinatario prima della transazione;
- **NewBalanceDest**: È un numero reale e contiene il patrimonio dell'acconto destinatario dopo la transazione;
- **IsFraud**: È un valore booleano (0, 1) e definisce se la transazione sia una frode (1) o no (0);

- **IsFlaggedFraud**: È un valore booleano (0, 1) e definisce se la transazione è stata rilevata come frode all'interno del sistema iniziale da cui proviene l'input.

Capitolo 2.1: Preprocessing del Dataset:

Dopo un'analisi del dataset, dimostra la presenza di alcune operazioni propedeutiche necessarie ai fini di rendere il dataset pronto per il sistema e i modelli a cui verrà affidati.

Si descrivono in seguito i diversi step di preprocessing effettuati:

1) È stata eliminata la colonna "**IsFlaggedFraud**" poiché non utile ai fini della predizione.
   Nel nostro sistema non c'è alcuna necessità di tenere conto che il sistema iniziale abbia "flaggato" che una transazione sia frode o meno poiché sarà il compito del classificatore verso i nuovi elementi;
2) È stata rilevata una relazione tra i valori delle colonne "**OldBalanceDest**", "**NewBalanceDest**", "**Amount**" e la colonna "**IsFraud**".
   Tale relazione è data dal fatto che, secondo lo studio del dataset, nei casi in cui le prime 2 colonne citate siano 0 ma la terza non lo sia, allora la transazione è più probabile che sia una frode.
   Ciò però non rende tale transazione una frode certa ed ognuna di tali transazioni è invece ambigua e deve seguire un controllo ulteriore più profondo per esser determinato che sia genuina o frode.
   Poiché non possiamo assegnare ad ogni riga che presenta tali valori la presupposizione che sia una frode, invece andiamo ad effettuare una modifica al significato di "IsFraud":
   a. **IsFraud = 0** se la transazione è genuina;
   b. **IsFraud = 1** se la transazione è ambigua;
   c. **IsFraud = 2** se la transazione è fraudolenta;
   d. Con tale modifica al significato di IsFraud, modifichiamo anche il dataset di conseguenza.
3) Tutte le colonne i cui valori sono stringhe non sono compatibili con i modelli di predizione che andremo ad utilizzare in seguito. Per questo motivo, la colonna "Type" è stata codificata e resa un valore numerico

tramite l'utilizzo di un dizionario, nominandola "**Type_Format**" ma lasciando presente "Type" all'interno del dataset per operazioni differenti.

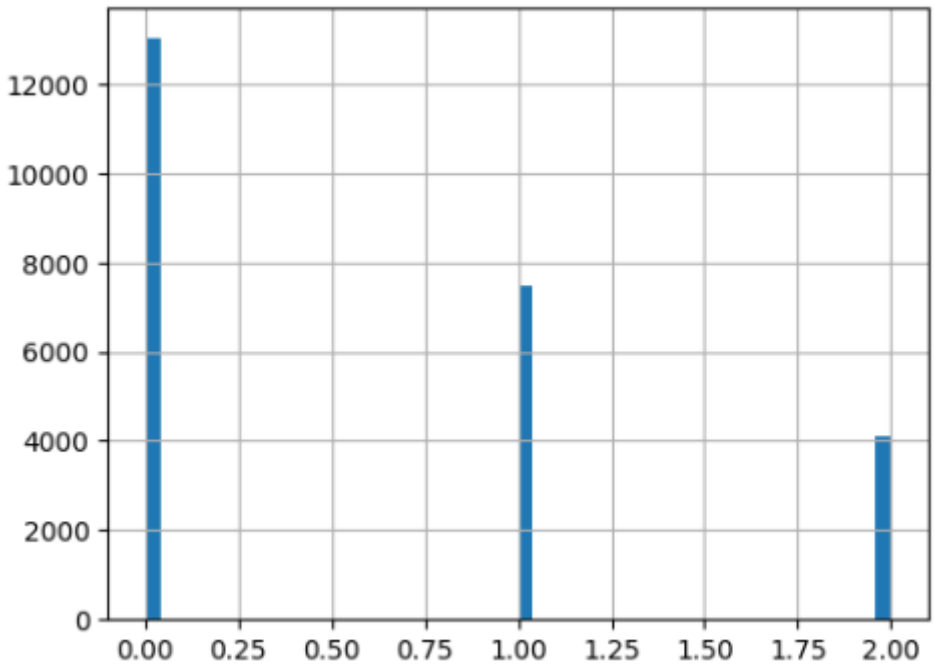
4) Poiché il dataset presenta un numero troppo elevato di elementi, è stato effettuato uno slicing di esso in modo tale da prendere solo un subset. Ciò è utile poiché non è necessario che il modello venga addestrato sull'intero dataset, bensì basta un subset equivalente di esso.
5) Il dataset in input presenta uno sbilanciamento tra le tre classi selezionate, il che poteva scatenare overfitting (di cui parleremo nel capitolo di apprendimento supervisionato) nella fase di addestramento del modello.
   Poiché è evidente che la presenza di sbilanciamento possa condurre ad overfitting, allora quest'ottimizzazione è stata fatta a priori.
   Per risolvere tale problema esistono principalmente tre vie:
   a. L'assegnamento di una costante che renda più significativo la presenza della classe minoritaria;
   b. Undersampling delle classi maggioritarie, prendendone solo una parte e scartando il resto cercando di avere, in proporzione, un subset veritiero dell'insieme iniziale;
   c. Oversampling delle classi minoritarie;
   d. La soluzione adottata è l'operazione di **Undersampling** poiché ci permette di rendere ancora più leggero il dataset.
      La classe minoritaria (t. fraudolente) ora è il 20% della somma delle due classi maggiori (66% t. genuine 33% t. ambigue), piuttosto che l'iniziale 1%.
      Dagli iniziali 650M elementi nel dataset, si termina il processo di preprocessing con 24636 elementi totali.

In seguito, invece, si descrivono gli step di preprocessing presi in considerazione ma scartati:

- È stata inizialmente presa in considerazione la possibilità di effettuare un'operazione di **target encoding** sui valori di tali colonne così da assegnare, piuttosto che il nome dell'origine/destinazione, la media di transazioni a cui partecipano in cui hanno effettuato frodi; effettivamente discretizzando i valori e rendendo possibile il loro input nel modello di predizione e di clustering.
  Tale opzione è stata scartata ed è stato scelto di non utilizzare tali colonne

nella predizione, così da evitare la presenza di **overfitting** data dal target encoding, che porterebbe ad una varianza troppo elevata.

Il rapporto tra le classi è il seguente:



Distribuzione delle classi nel dataset

Capitolo 3: Apprendimento Supervisionato

L'apprendimento supervisionato è un ramo dell'apprendimento automatico all'interno di cui un modello viene addestrato in base ad un insieme di dati in input con **etichette**. Esistono due tipi di apprendimenti supervisionato:

- **Classificazione**: I possibili valori delle etichette appartengono ad un dominio finito numerico. Ciò significa che l'etichetta può avere un valore x € I dove I è un insieme enumerativo discreto.
  Un esempio è la classificazione booleana all'interno di cui l'etichetta può avere valore *true* o *false*;
- **Regressione**: All'interno di cui l'etichetta può assumere qualsiasi valore reale. Il dominio a cui appartiene l'etichetta è un dominio numerico continuo.

L'Apprendimento supervisionato ha il compito di definire delle relazioni tra i valori dell'insieme e l'etichetta ad essi associata.
Successivamente, al termine dell'apprendimento, il modello avrà l'obiettivo di esercitare la sua conoscenza ai fini di predire quale sia l'etichetta di tutti quei **nuovi dati** che non ne presentano una, in base al valore di tali dati.

Al fine di poter addestrare il modello più adatto al nostro dataset, è necessario seguire i seguenti passi:

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

Tale processo dovrà essere eseguito per ogni strategia di rilevazione degli iper-parametri che vogliamo ammettere, e per ogni modello che vogliamo valutare.

La tipologia di apprendimento supervisionato selezionata è la Classificazione, poiché il sistema prevede un insieme finito di 3 classi possibili per ogni esempio target.
I modelli di apprendimento supervisionato valutati sono i seguenti:

- **K-NN** (K-Nearest Neighbours): Il KNN è un modello di classificazione che non effettua apprendimento nel senso tradizionale, e che invece per

predire i valori target (etichette) utilizza delle metriche come la moda, la media o un'interpolazione dei valori target dei k esempi dati.
L'algoritmo va a definire i k elementi più simili al nuovo esempio da predire, e assegna ad esso la classe più comune tra tali k elementi;

- **Random Forest**: Modello di classificazione fondato sulla media delle decisioni di alberi decisionali.
  Il modello, infatti, crea un numero n di alberi decisionali ed effettua la predizione dell'etichetta del nuovo elemento con ognuno di essi, dopodiché sceglie la sua classe in base alla media delle predizioni di ogni albero appartenente alla "foresta";
- **Multinomial Naive Bayes**: Classificatore basato sul teorema di Bayes utilizzato per predire dati discreti.
  Il Naive Bayes presuppone che le features dell'insieme di apprendimento siano indipendenti l'una dall'altra e che quindi non ci sia alcuna dipendenza funzionale tra di loro (es. A -> B).
  Esistono diversi tipi di algoritmi di Naive Bayes, ed è stato selezionato il Multinomial poiché la predizione viene effettuata su un dominio di classi contenente 3 elementi.
- **Regressione Logistica Multinomiale**: Modello di classificazione basato sulla regressione, che normalmente predice valori continui, ma che in questo caso la regressione va a calcolare la probabilità che un esempio appartenga a una certa classe.

Capitolo 3.1: Ottimizzazione degli Iper-Parametri

I modelli di apprendimento automatico fanno uso degli Iper-Parametri per specificare i dettagli del processo di apprendimento, ed influenzano il processo di addestramento senza poter essere appresi direttamente dai dati.
Per questo motivo si effettua la fase di ottimizzazione degli Iper-Parametri.

La ricerca degli Iper-Parametri consiste nel trovare, in base ad un insieme di iper-parametri di un modello possibili, quali di essi sono i più adatti al dataset e al contesto del nostro sistema.
Nel nostro caso ne utilizzeremo di molteplici, valutando solamente gli iper-parametri che permettono risultati più efficienti.

Gli algoritmi per la scelta degli Iper-Parametri sono:

- **Bayes Search**: Si effettua un'ottimizzazione degli iper-parametri effettuando utilizzando la modellazione bayesiana, basata sulle teorie delle probabilità bayesiane.
  Utilizza una distribuzione di probabilità a priori per rappresentare le credenze iniziali sugli iper parametri, e poi la aggiorna in base alle nuove evidenze osservate; ciò significa che il processo di ottimizzazione migliora ad ogni sua iterazione;
- **Grid Search**: Tecnica che, per ogni iper parametro, fornisce una griglia di valori e testa ogni possibile combinazione di valori.
  Questa operazione però è molto costosa nel caso in cui gli iper parametri siano molti, e ha complessità temporale O(n^k) dove:
  - n = numero di iper parametri;
  - k = possibili valori degli iper parametri.
  - n^k = ogni possibile combinazione.
- **Random Search**: che effettua il campionamento in maniera simile a quella della Grid Search, ma che invece di esplorare tutte le possibili combinazioni ne prende solamente un sottoinsieme di esse fino a quando non viene trovata una combinazione che superi una certa soglia di efficienza.
  La Random Search verrà utilizzata nei casi dei modelli in cui la ricerca a griglia sia troppo dispendiosa dati gli iper parametri, grandezza del dataset e complessità temporale del modello selezionato.

Ad ognuna di esse è stata applicata la tecnica di K-Fold Cross Validation in modo tale da massimizzare la generalizzazione del modello di predizione.
Tale tecnica permette di dividere l'insieme utilizzato per l'addestramento in k sottoinsiemi. Si effettua successivamente l'addestramento su k-1 fold e viene utilizzato l'ultimo (k) per la valutazione del modello.
La costante k selezionata è 10, poiché è il valore di default nel caso in cui il dataset utilizzato presenta più di 10000 elementi.

Per ogni modello è stato selezionato un insieme di iper-parametri tra cui selezionare gli ottimali, in base a delle convenzioni basate sull'analisi delle prestazioni e in base al dataset/modello.

Sui risultati sono stati valutati, inoltre, i migliori iper parametri ottenuti.
Nei casi in cui Random Forest e Grid Search appaiono entrambi è stato scelto di ignorare i risultati di Random Forest, poiché il Grid ricerca su più combinazioni.

K-NN:

- **Number of Neighbours**: Che definisce il valore "k" di vicini selezionati per la predizione della classe. I valori testati sono: [5, 7, 9, 13];
- **Weights**: La funzione di peso utilizzata per la previsione sui k vicini. I valori testati sono: ['uniform', 'distance'].
    - Uniform: tutti i vicini hanno peso uniforme;
    - Distance: i vicini più vicini hanno anche peso maggiore. I più lontani hanno peso minore;
- **Metric**: Metrica definita per la computazione della distanza. I valori testati sono: ['Minkowski', 'Manhattan']
    - Minkowski: Viene utilizzata la distanza euclidea;
    - Manhattan: Viene utilizzata la distanza di Manhattan;

```python
# Ricerca dei best params
hyperparameters_knn = {
    "n_neighbors": (5, 7, 9, 13),
    "weights": ("uniform", "distance"),
    "metric": ("minkowski", "manhattan")}
```

Risultati:

```
Best params Grid {'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'distance'}
Best params Bayes OrderedDict({'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'distance'})
Best params Random {'weights': 'distance', 'n_neighbors': 5, 'metric': 'manhattan'}
```

Random Forest:

- **Number of estimators**: definisce il numero di alberi all'interno della foresta, su cui faremo la media. I valori testati sono: [100, 500, 800, 1200]
- **Max Depth**: profondità massima degli alberi decisionali. I valori testati sono: [15, 25, 35];
- **Criterion**: Determina la qualità dello split effettuato sui nodi. I valori testati sono: ['gini', 'entropy']
    - Gini: Va a misurare quanto spesso un elemento viene classificato con una classe errata;

    - Entropy: Si ispira al concetto di entropia per definire la quantità di ordine all'interno dei dati. Se l'entropia è alta, allora c'è perdita di informazioni. Entropia bassa, viceversa, porta a guadagno di informazioni;
- **Min samples split**: Un criterio di split viene inserito solo dopo che viene raggiunto il numero minimo di campioni nello split definito da questo iper-parametro. I valori testati sono: [2, 5, 10];
- **Min Samples Leaf**: Numero minimo di campioni richiesti in un nodo foglia. I valori testati sono: [1, 2, 4].

```
hyperparameters_rf = {
    'n_estimators': (100, 500, 800, 1200),
    'max_depth': (15, 25, 35),
    'min_samples_split': (2, 5, 10),
    'min_samples_leaf': (1, 2, 4)
}
```

Risultati:

```
Best params Bayes OrderedDict({'max_depth': 25, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 1200})
Best params Random {'n_estimators': 800, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_depth': 25}
```

Logistic Regression:

- **C Values**: Valori di inversione della forza di regolarizzazione. In base al valore di C è definita la rigidità del modello. Un modello rigido (C basso) tende meno all'overfitting ma potrebbe portare a underfitting. Un modello flessibile (C alto) potrebbe adattarsi troppo ai dati di addestramento (overfitting) ma è utile nel caso di presenza di rumore. I valori utilizzati sono: [100, 10, 1.0, 0.1, 0.01]
- **Penalty**: Definisce la forma di normalizzazione nella somma dei quadrati dei pesi. È stato definito 'l2' come unico poiché permette pesi più bilanciati e previene overfitting e perché è l'unico supportato da due dei solvers presi in considerazione;
- **Solvers**: Algoritmo utilizzato nel problema di ottimizzazione. I valori testati sono: ['newton-cg', 'lbfgs', 'liblinear']
    - Newton-cg: usa un metodo di ottimizzazione basato sulle derivate seconde, utile per problemi di grandi dimensioni ma supporta solo la regolarizzazione L2;

- Lbfgs: Sta per “Limited-Memory Broyden-Fletcher-Goldfarb-Shanno” e approssima un metodo basato sulle derivate seconde ma per supportare problemi di piccole o medie dimensioni ma supporta solo la regolarizzazione L2;
- Liblinear: Usa un algoritmo di coordinata lineare ed è efficace nei problemi di piccole dimensioni, e supporta sia regolarizzazione l1 che l2.

```
hyperparameters_clf = {
    'C' : (100, 10, 1.0, 0.1, 0.01),
    'max_iter' : (25, 50, 100)
}
```

Risultati:

```
Best params Grid {'C': 100, 'max_iter': 25}
Best params Bayes OrderedDict({'C': 100.0, 'max_iter': 100})
```

Multinomial Naive Bayes:

- Alpha: Parametro di smoothing additivo. Un valore > 0 aiuta a gestire il problema dei conteggi 0 per alcune caratteristiche del dataset. I valori testati sono: [0.01, 0.1, 1, 10, 100];
- Fit Prior: Determina se imparare o meno le probabilità a priori delle classi. Può avere valore:
  - True: per imparare le probabilità di classe col tempo in base alla frequenza con cui appaiono nel dataset;
  - False: l’algoritmo usa una distribuzione uniforme per le probabilità a priori delle classi, quindi contando come ugualmente probabili le classi prima di osservare i dati.

```
hyperparameters_multinomial_bayes = {
    'alpha': (0.01, 0.1, 1, 10, 100),
    'fit_prior': (True, False)}
```

Risultati:

```
Best params Grid {'MultinomialNB__alpha': 100, 'MultinomialNB__fit_prior': False}
Best params Bayes OrderedDict({'MultinomialNB__alpha': 100.0, 'MultinomialNB__fit_prior': False})
Best params Random {'MultinomialNB__fit_prior': False, 'MultinomialNB__alpha': 100}
```

Capitolo 3.2: Addestramento e testing del Modello

Data la grandezza del dataset, non è stato ritenuto necessario effettuare la K-Fold Cross Validation durante la fase di addestramento.

Propedeuticamente all'implementazione dei modelli, è stato necessario dividere il dataset input in training set e test set in scala 70% - 30%.
Il modello è stato successivamente addestrato sul training set e valutato usando la strategia di **Cross Validation**, così da valutare le prestazioni del modello sia sul training set che sul test set, contenente valori nuovi al modello, a cui esso non si è adattato.

Allego un esempio di implementazione del codice, pronto già alla valutazione del modello:

```
def logistic_regression_classification(training, target):
    x_train, x_test, y_train, y_test = train_test_split(training, target,
test_size=0.3, random_state=42, shuffle=True)
    clf = LogisticRegression(multi_class='auto', random_state=42,
penalty='l2', solver='newton-cg', C=100, max_iter=100)
    clf.fit(x_train, y_train)
    y_pred_rf = clf.predict(x_test)
    print("Accuracy clf test set:", metrics.accuracy_score(y_test,
y_pred_rf))

    print(metrics.classification_report(y_test, y_pred_rf))

    y_pred = clf.predict(x_train)
    print("Accuracy clf training set:", metrics.accuracy_score(y_train,
y_pred))
    print(metrics.classification_report(y_train, y_pred))
    plot_learning_curves(clf, training.to_numpy(), target.to_numpy(),
"Logistic Regression")

    return clf
```

Capitolo 3.3: Valutazione delle prestazioni

Una volta determinati gli iper-parametri per ognuno dei modelli, è possibile procedere con la valutazione delle prestazioni, in modo tale da definire quale modello verrà utilizzato nel progetto finale.

Nella valutazione delle performance vengono presi in atto i valori di:

- **Accuracy**: Misura della correttezza del modello, ed è definita dal numero di previsioni corrette rispetto al totale delle previsioni.
    - Formula: $\frac{|predizioni\ corrette\ totali|}{|predizioni\ totali|}$
- **Precision**: Definisce la totalità delle previsioni effettuate con successo per ogni classe del target.
    - Formula: $\frac{|predizioni\ corrette\ classe\ c|}{|predizioni\ totali\ classe\ c|}$
- **Recall**: Definisce la totalità delle istanze di classe c classificate nella classe c in rapporto con il numero di istanze nella classe c.
    - Formula: $\frac{|istanze\ predette\ come\ classe\ c|}{|istanze\ classe\ c|}$
- **F1**: Media armonica tra precision e recall
    - Formula: $\frac{2*P*R}{P+R}$

Viene, inoltre, presa in considerazione la curva di apprendimento per la rilevazione di overfitting/underfitting all'interno dei modelli valutati.

In seguito sono presenti i risultati per ogni modello nella predizione sul training e test set.

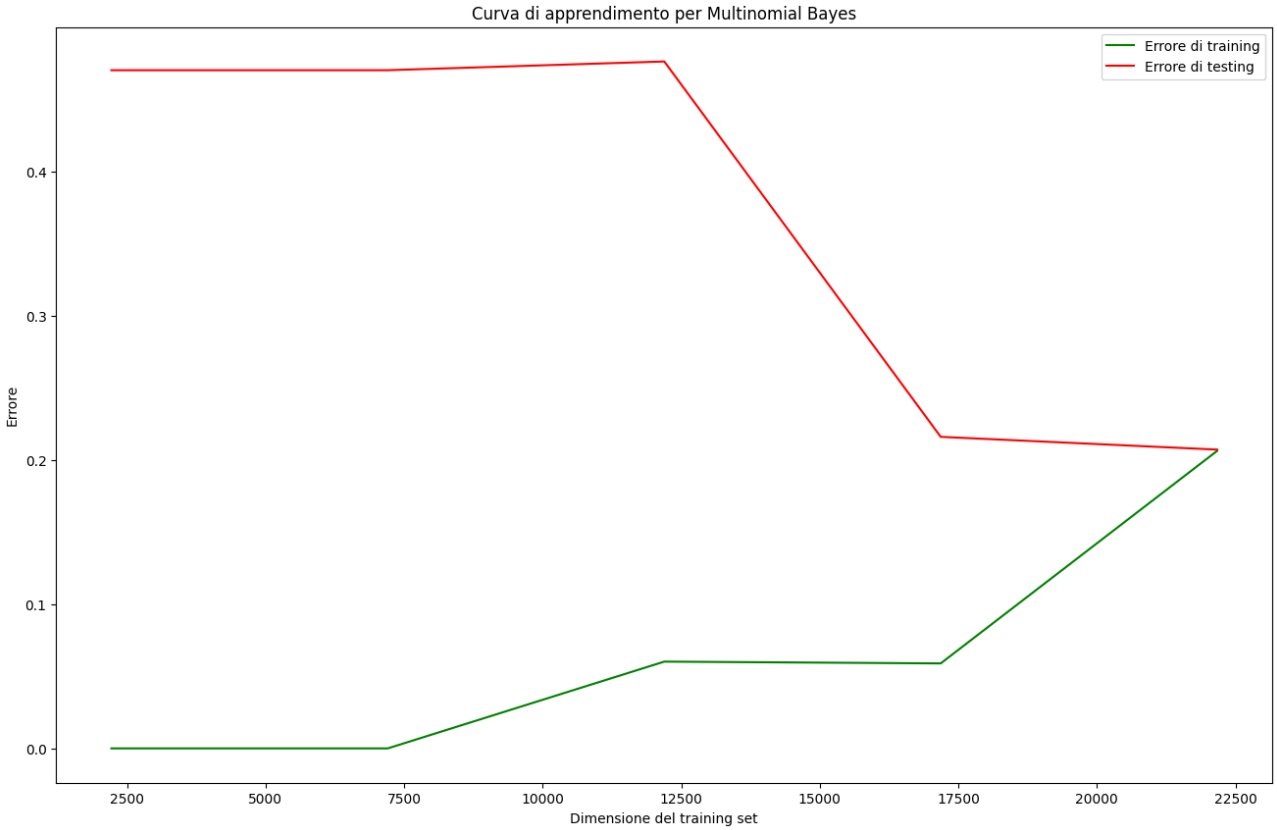
Multinomial Naive Bayes:

Prestazioni dei parametri di valutazione Multinomial Naive Bayes:

```
Multinomial Naive Bayes:
Accuracy multinomial bayes test set: 0.9545393045595996
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      3933
           1       0.93      0.93      0.93      2219
           2       0.86      0.87      0.87      1239

    accuracy                           0.95      7391
   macro avg       0.93      0.93      0.93      7391
weighted avg       0.96      0.95      0.95      7391

Accuracy multinomial bayes training set: 0.9523312456506611
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      9123
           1       0.93      0.93      0.93      5254
           2       0.85      0.87      0.86      2867

    accuracy                           0.95     17244
   macro avg       0.93      0.93      0.93     17244
weighted avg       0.95      0.95      0.95     17244
```

Prestazioni del Multinomial Naive Bayes

Curva di apprendimento Multinomial Naive Bayes:



Learning Curve del Multinomial Naive Bayes

KNN:

Prestazioni dei parametri di valutazione K-NN:

Valori di Deviazione e di Varianza:

- Deviazione standard dell'errore di addestramento: 0.010793276956730102;
- Deviazione standard dell'errore di test: 0.014324459468623608;
- Varianza dell'errore di addestramento: 0.00011649482746468103;
- Varianza dell'errore di test: 0.00020519013906824054

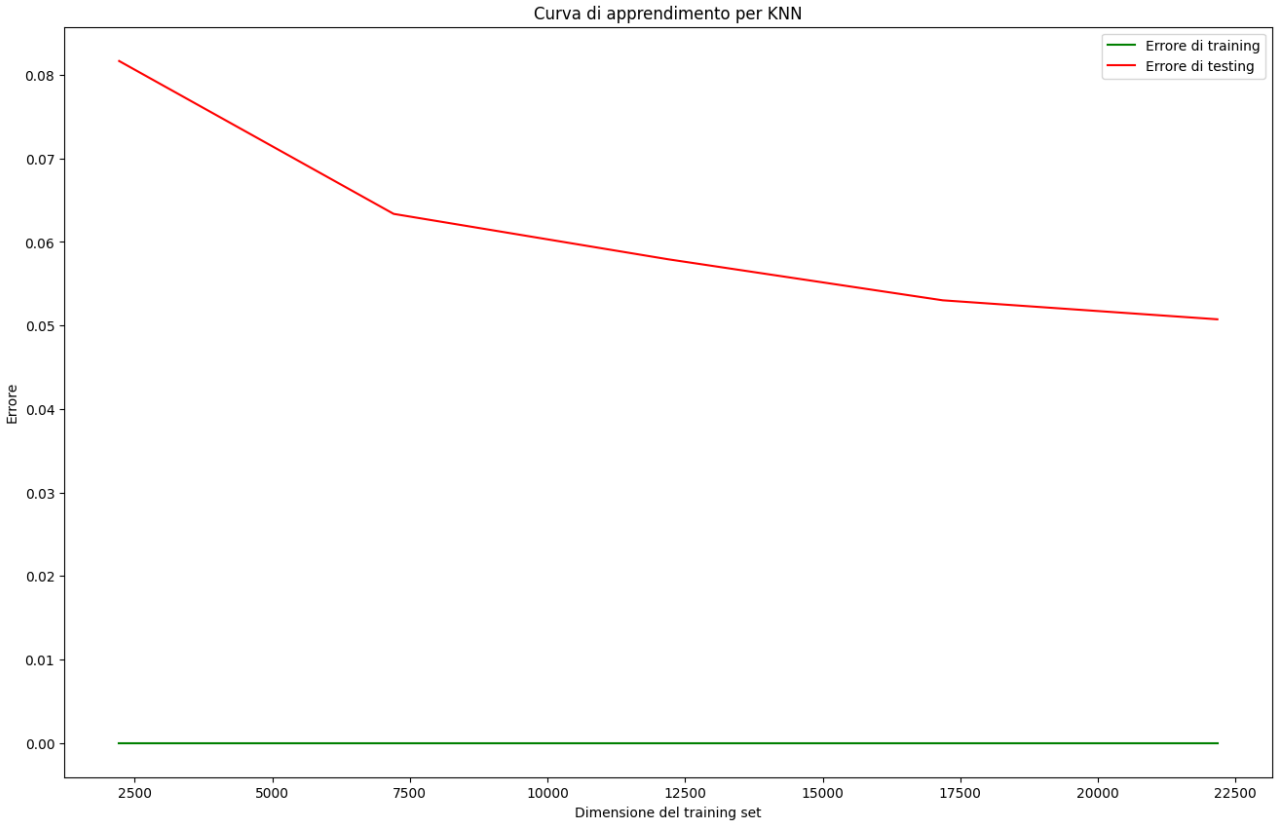
```
KNN:
Accuracy knn test set: 0.9452036260316601
              precision    recall  f1-score   support

           0       0.95      0.95      0.95      3968
           1       0.97      0.98      0.97      2177
           2       0.89      0.88      0.88      1246

    accuracy                           0.95      7391
   macro avg       0.94      0.94      0.94      7391
weighted avg       0.95      0.95      0.95      7391

Accuracy knn training set: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      9088
           1       1.00      1.00      1.00      5296
           2       1.00      1.00      1.00      2860

    accuracy                           1.00     17244
   macro avg       1.00      1.00      1.00     17244
weighted avg       1.00      1.00      1.00     17244
```

Prestazioni del K-NN

Curva di apprendimento K-NN:

Valori di Deviazione e di Varianza:

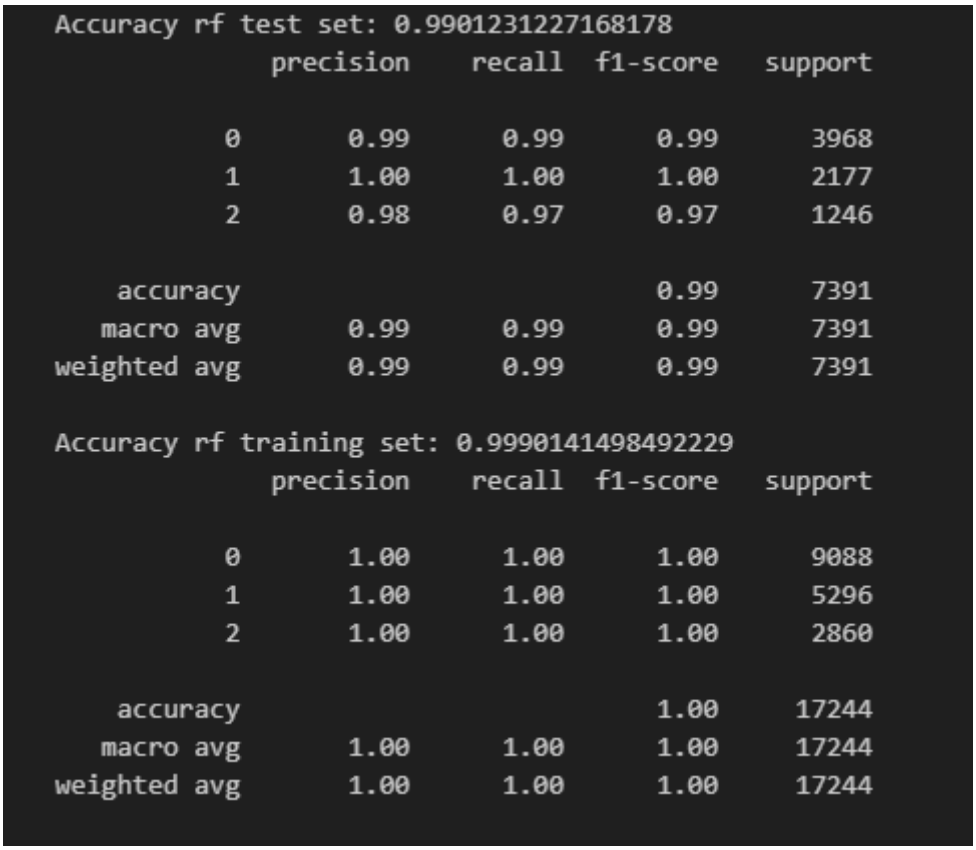
- Deviazione standard dell'errore di addestramento: 0.0;
- Deviazione standard dell'errore di test: 0.0032304298841907882;
- Varianza dell'errore di addestramento: 0.0;
- Varianza dell'errore di test: 1.043567723667291e-05



Learning Curve del K-NN

Random Forest:

Prestazioni dei parametri di valutazione Random Forest:



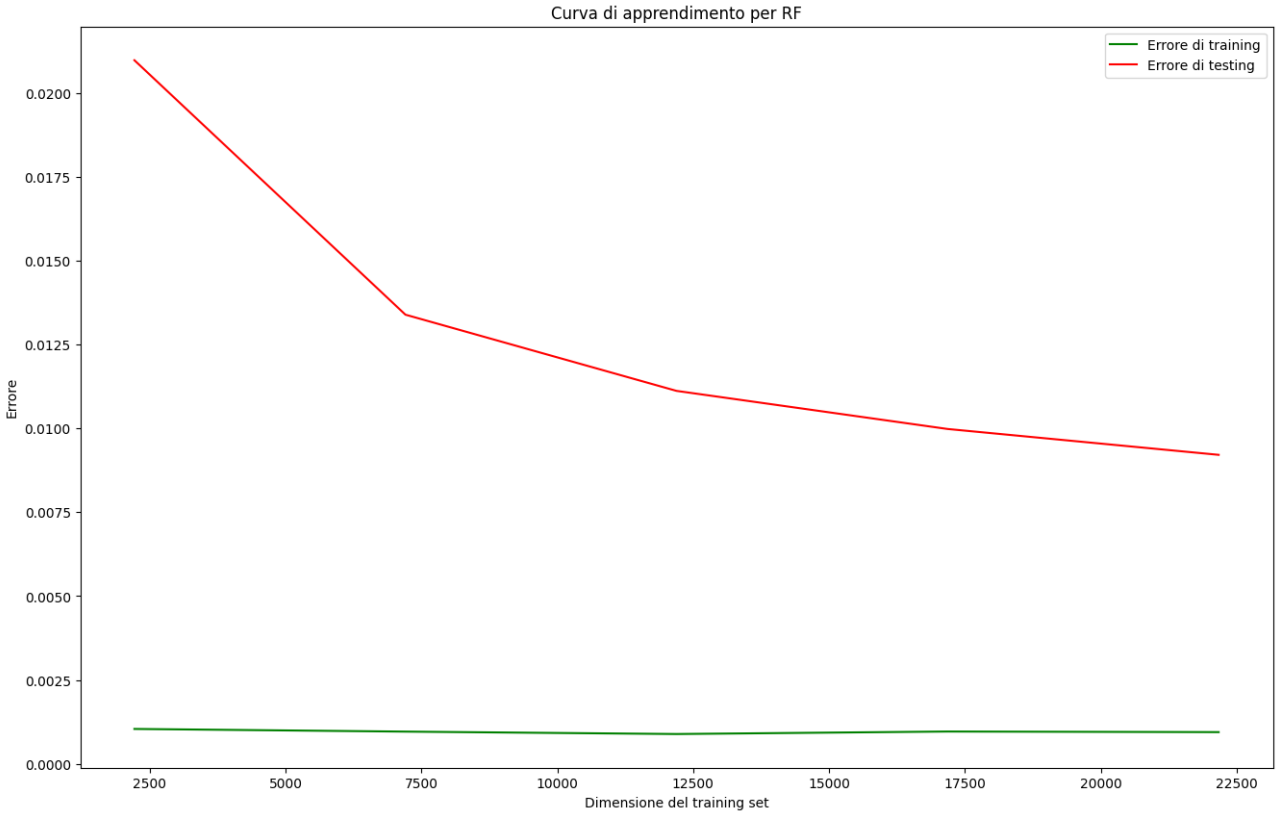
```
Accuracy rf test set: 0.9901231227168178
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3968
           1       1.00      1.00      1.00      2177
           2       0.98      0.97      0.97      1246

    accuracy                           0.99      7391
   macro avg       0.99      0.99      0.99      7391
weighted avg       0.99      0.99      0.99      7391

Accuracy rf training set: 0.9990141498492229
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      9088
           1       1.00      1.00      1.00      5296
           2       1.00      1.00      1.00      2860

    accuracy                           1.00     17244
   macro avg       1.00      1.00      1.00     17244
weighted avg       1.00      1.00      1.00     17244
```

Prestazioni del Random Forest

Curva di apprendimento Random Forest:

Valori di Deviazione e di Varianza:

- Deviazione standard dell'errore di addestramento: 7.92855343974022e-05;
- Deviazione standard dell'errore di test: 0.0015401783637876 89;
- Varianza dell'errore di addestramento: 6.286195964681646e-09;
- Varianza dell'errore di test: 2.3721493922797226e-06



Learning Curve del Random Forest

Logistic regression:

Prestazioni dei parametri di valutazione Logistic Regression:

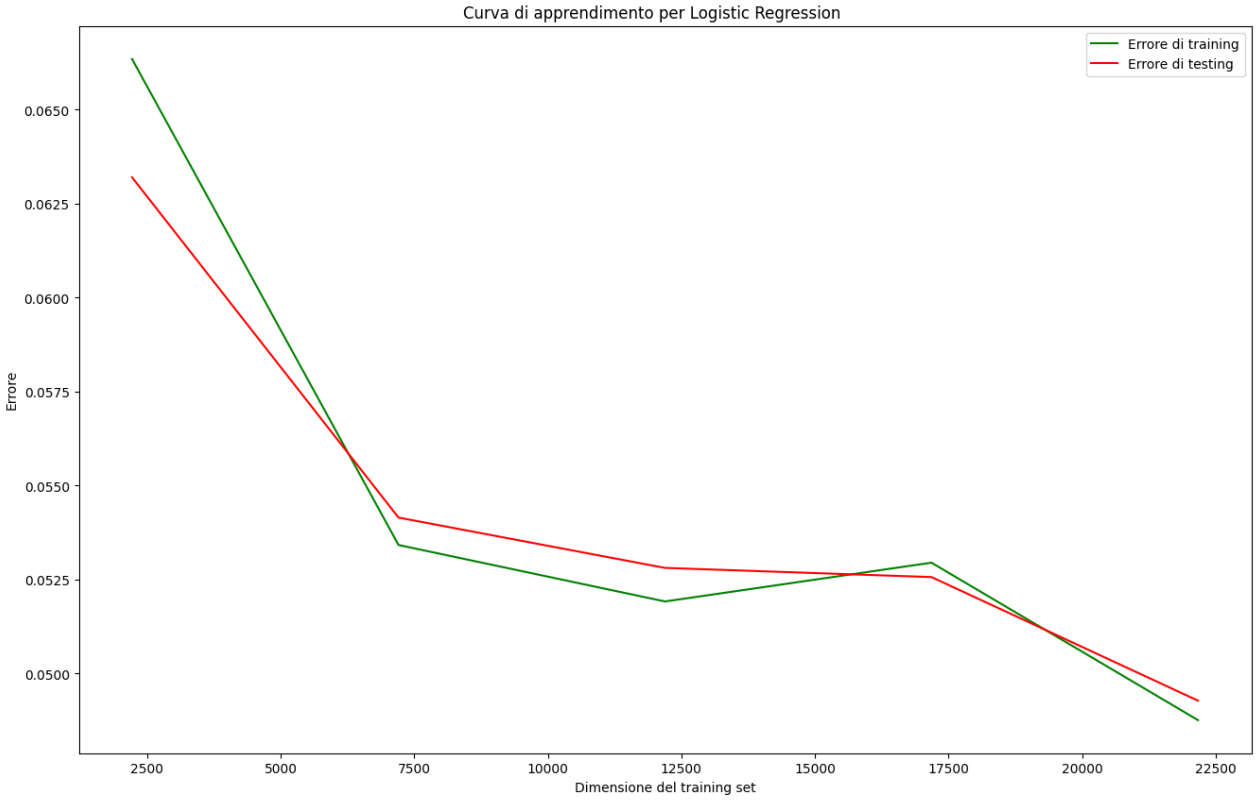
```
Logistic Regression:
Accuracy clf test set: 0.9470978216750101
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      3968
           1       0.96      0.98      0.97      2177
           2       0.95      0.83      0.88      1246

    accuracy                           0.95      7391
   macro avg       0.95      0.92      0.94      7391
weighted avg       0.95      0.95      0.95      7391

Accuracy clf training set: 0.9508814660171654
              precision    recall  f1-score   support

           0       0.94      0.97      0.96      9088
           1       0.97      0.99      0.98      5296
           2       0.94      0.83      0.88      2860

    accuracy                           0.95     17244
   macro avg       0.95      0.93      0.94     17244
weighted avg       0.95      0.95      0.95     17244
```

Prestazioni della Logistic Regression

Curva di apprendimento Logistic Regression:

Valori di Deviazione e di Varianza:

- Deviazione standard dell'errore di addestramento: 0.0026962552490869406;
- Deviazione standard dell'errore di test: 0.0028641984297047814;
- Varianza dell'errore di addestramento: 7.269792368228879e-06;
- Varianza dell'errore di test: 8.203632644723336e-06



Learning Curve della Logistic Regression

Capitolo 3.4: Analisi dei risultati:

Dai risultati possiamo notare che il K-NN, l'algoritmo di Logistic Regression e il Random Forest hanno tutti avuto risultati ottimi e coerenti in training e test set, mostrando risultati elevati in assenza di Overfitting.
Ciò possiamo inoltre denotarlo tramite la **curva di apprendimento e i valori di deviazione e varianza**, che mostrano l'andamento de modello in base al numero di elementi all'interno del set.

Valori bassi di deviazione standard e varianza all'interno del training set, infatti, mostrano che un modello si adatta bene ai dati di addestramento e che il modello rimane stabile e coerente durante l'intero addestramento.
Valori bassi di deviazione standard e varianza all'interno del test set, invece, mostrano che il modello generalizza bene i dati non visti e quindi dimostra la mancanza di overfitting.

L'**overfitting**, infatti, è un effetto collaterale dell'adattamento del modello ai dati, che consiste nel sovradattamento del modello al training set.
Ciò che accade è che il modello impara a memoria i dati del training, quindi è

molto bravo a predire elementi del training set ma è meno abile a riconoscere i pattern, e quindi a classificare, di dati nuovi al modello.

Notiamo invece, nel caso del Multinomial Naive Bayes, che il modello soffre di Underfitting e presenta dei valori di deviazione standard 10 volte più alti degli altri modelli, e di varianza 100 volte più alti degli altri modelli.

L'**underfitting** si distingue dall'overfitting perché, al contrario, il modello non è capace di adattarsi ai dati e quindi ottiene risultati casuali, tipicamente bassi sia nel training che nel test set.

Ciò potrebbe essere causato da diversi fattori, come la presenza di rumore, scarsità di dati, modello troppo semplice.
In questo caso, però, deriva da una **mancanza di regolarizzazione adeguata**.

Risoluzione Underfitting Multinomial Naive Bayes:

A differenza degli altri modelli, il Multinomial Naive Bayes è molto dipendente dai suoi Iper-Parametri, ed evidentemente nel dominio iniziale non c'erano Iper-Parametri tali da permettere al modello di adattarsi bene ai dati.
In questo caso ci si riferisce all'iper-parametro "alpha", che definisce la regolarizzazione.

Si rieffettua la ricerca degli iper-parametri usando valori di alpha maggiori e settando "fit_prior" a False poiché siamo già certi che sia valido:

```
hyperparameters_multinomial_bayes = {
    'MultinomialNB__alpha': (100, 300, 400, 500, 700, 900)}
p = Pipeline([('Normalizing',MinMaxScaler()),('MultinomialNB',MultinomialNB(fit_prior=False))])
```

Ne otteniamo i seguenti iper-parametri. Sia il Bayes search che il Grid Search hanno ottenuto gli stessi risultati quindi procederemo con alpha=300.

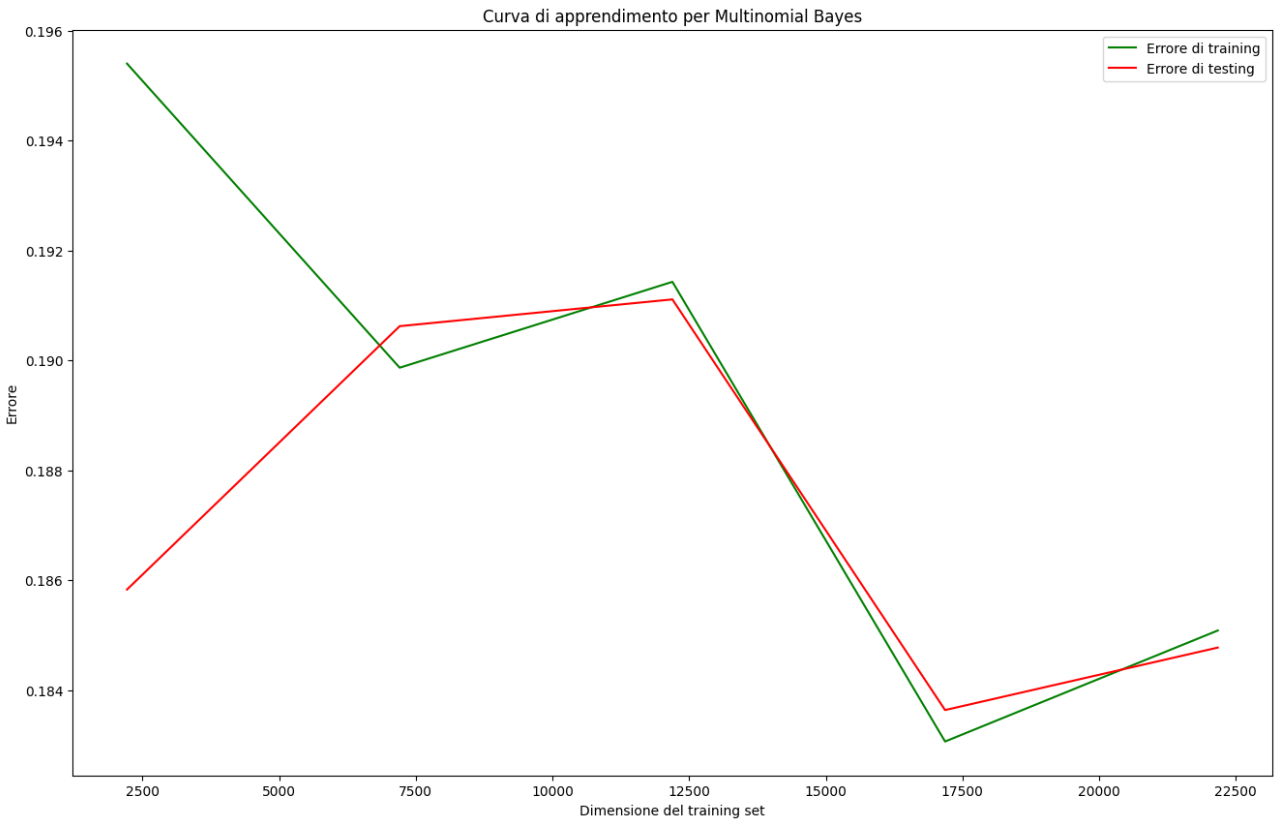
```
Multinomial Naive Bayes BEST PARAMS:
Fitting 10 folds for each of 6 candidates, totalling 60 fits
Best params Grid {'MultinomialNB__alpha': 300}
Best params Bayes OrderedDict({'MultinomialNB__alpha': 300})
```

Otteniamo i seguenti nuovi risultati riguardi i parametri di valutazione:

```
Multinomial Naive Bayes:
Accuracy multinomial bayes test set: 0.8140982275740766
              precision    recall  f1-score   support

           0       0.78      0.98      0.87      3968
           1       0.88      0.93      0.90      2177
           2       0.81      0.10      0.18      1246

    accuracy                           0.81      7391
   macro avg       0.83      0.67      0.65      7391
weighted avg       0.82      0.81      0.76      7391

Accuracy multinomial bayes training set: 0.8150081187659476
              precision    recall  f1-score   support

           0       0.78      0.98      0.87      9088
           1       0.89      0.93      0.91      5296
           2       0.80      0.10      0.17      2860

    accuracy                           0.82     17244
   macro avg       0.82      0.67      0.65     17244
weighted avg       0.82      0.82      0.76     17244
```

E la seguente curva di apprendimento e parametri di deviazione e varianza:

- Deviazione standard dell'errore di addestramento: 0.0005884295794434491;
- Deviazione standard dell'errore di test: 0.0034693940830227183;
- Varianza dell'errore di addestramento: 3.4624936996399433e-07;
- Varianza dell'errore di test: 1.2036695303313048e-05



Learning Curve del Multinomial Naïve Bayes post-risoluzione

Notiamo dei miglioramenti sul modello in tutti i termini di valutazione.

Dai risultati, che potremmo ottenere risultati migliori, ma non sarà necessario perché sarebbero comunque inferiori ai risultati degli altri tre modelli.

Decisione del Modello finale:

Dai risultati possiamo notare che il Random Forest è il modello che ha portato i risultati migliori, vantando dei parametri di valutazione più alti e di una Learning Curve coerente con mancanza di overfitting, valori di deviazione standard e varianza in training e test set bassi, e il fatto che a differenza del K-NN effettui apprendimento in senso tradizionale.

Capitolo 4: Inclusione del Large Language Model

Questa fase si basa sull'inclusione in fase di preprocessing dell'etichetta "IsFraud = 2" che definisce che una transazione sia "Ambigua".

In fase di esecuzione, dopo l'inserimento e la classificazione della transazione in input, è possibile ricevere come output della predizione, l'etichetta

“Ambigua”; essa necessiterebbe di personale per il controllo della transazione, in modo tale da inserirla come frode o genuina.

È possibile ottimizzare il lavoro che un personale umano effettua sulla transazione tramite l’utilizzo di un Large Language Model per l’analisi delle transazioni ambigue, così che definisca se tale transazione sia infine genuina o fraudolenta.

Capitolo 4.1: Introduzione ai Large Language Models

La modellazione del linguaggio umano su larga scala è un processo complesso che comprende la creazione di modelli linguistici che effettuano **predizioni** riguardo un testo dato in input al modello.

Il modello linguistico ha il compito di predire quali sequenze di testo, in relazione alla sua base di conoscenza, siano i più adatti al set di dati in ingresso.

La modellazione del linguaggio si forma sulle basi dei Transformers, ovvero architetture progettate sull’idea dell’*attenzione*, che già prima dei LLM venivano utilizzati nella traduzione dei testi tramite transformers traduttori.
Il transformer ha il compito di ricevere in ingresso un set di dati, codificarlo in un linguaggio intermedio, formato da **tokens** in base alle sillabe delle parole, tra macchina e umano; dopodiché effettua operazioni su di esso e poi lo decodifica nuovamente in modo tale da poter essere compreso dall’utente.

L’attenzione di un transformer è verso i dati e verso sé stesso.
Quando l’attenzione viene riposta verso sé stessi, il transformer si chiede domande del tipo “Quanto è importante ogni altro token di input per **me**, il transformer?”
In base all’importanza di appartenenza di ogni parola in ingresso verso il resto delle altre parole in ingresso, il transformer va a definire quale sia l’importanza di un’altra parola che potrebbe includere, prima di includerla; così include le parole che hanno **importanza maggiore** secondo la base di conoscenza appresa durante l’addestramento del LLM.

Capitolo 4.2: Feed Forward Neural Network nei Transformers

L’architettura del modello base di un Transformer di text generation, che ha il ruolo di decoder di token -> parole, prevede l’utilizzo di due strati principali:

- Self-attention layer;

- Feed Forward Network (FFN).

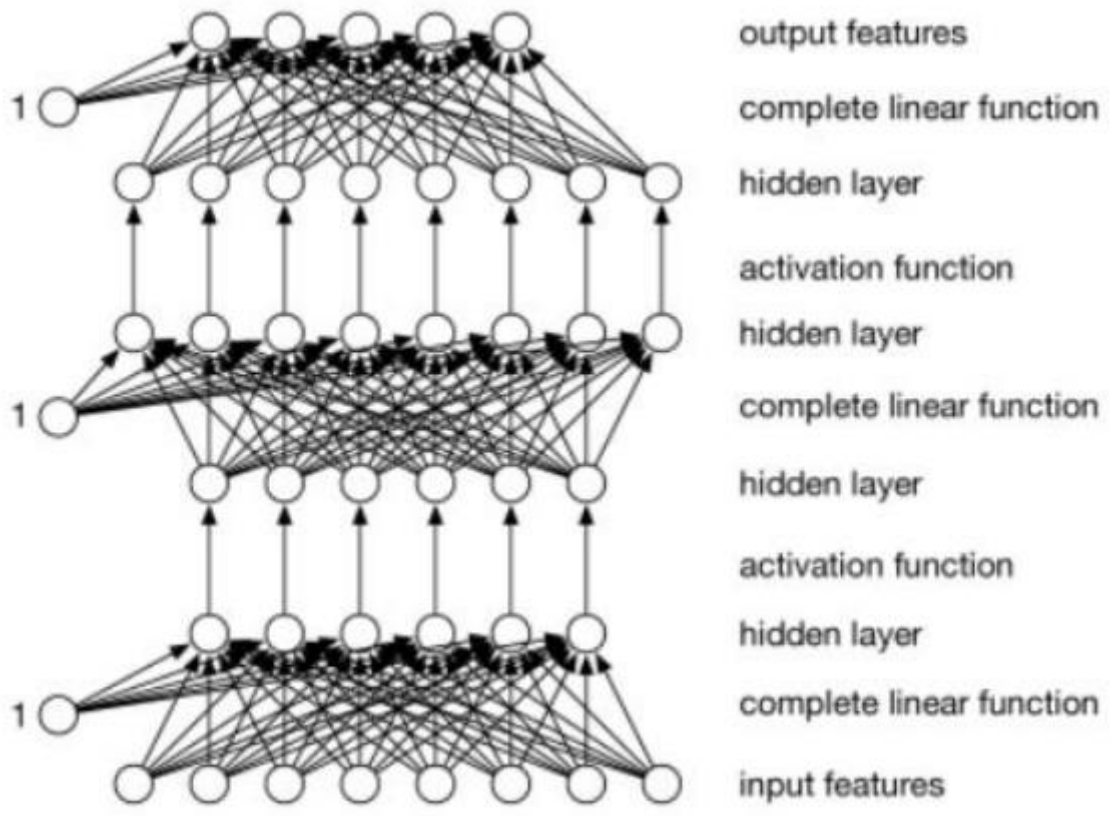
Il Self-attention layer è atto alla definizione dell'importanza dei token.

Il Feed-Forward Neural Network è un modello utilizzato per problemi di ragionamento di basso livello con molti dati disponibili, che si ambientano perfettamente nel Decoder di un Transformer poiché lavorano su un numero molto ampio di token: ragionamento di basso livello.

Le FFN sono viste come una gerarchia di funzioni lineari, frapposte a funzioni di attivazione. Presentano features di input e feature di output.
Le features di input permettono di alimentare degli strati nascosti all'interno della rete neurale: funzioni che combinano output di unità degli strati precedenti, per poi alimentare infine le predizioni degli output.

I layer di una FFN possono essere di tre tipi:

- **Layer di input**: Un'unità di input viene generata per ogni feature in input per gestire tale feature;
- **Layer completo lineare**: Tramite una funzione lineare dei valori in input allo strato e il peso da apprendere, l'output del layer completo lineare è il prodotto tra il valore con il peso appreso a lui associato.
- **Layer di attivazione**: L'output del layer di attivazione è calcolato tramite una funzione di attivazione in corrispondenza al valore di input. È tipicamente una funzione sigmoidale o la ReLU: Rectified Lineare Unit. La ReLU è definita come: $f(x) = \max(0, x)$ dove x è il valore in input.



Architettura di una Feed Forward Neural Network

FFN è una qualsiasi feed forward neural network con una funzione di attivazione non lineare (come la ReLU). Ogni strato di nodi, ricevendo input dallo strato precedente, applica una funzione di attivazione non lineare per passare poi l'output allo strato successivo.
La non linearità permette di apprendere relazioni complesse tra input ed output.

L'output dello strato del self-attention viene passato al primo strato lineare del layer FFN, dopodiché viene applicata la funzione di attivazione non lineare all'output del primo strato lineare, che poi lo passa al secondo strato lineare del layer FFN.

Capitolo 4.3: Implementazione del Large Language Model

Per poter comunicare con un Large Language Model c'è bisogno di creare una connessione con esso.
Nel nostro caso, utilizzeremo prima di tutto il modello "OpenAI", che fornisce API per l'inclusione del linguaggio nel nostro codice Python.

Creiamo la connessione tramite il seguente codice:

```
    response = openai.ChatCompletion.create(
        model = MODEL,
        messages=messages,
        temperature=0,
    )
```

Il codice svolge quattro compiti principali:

- Definisce il modello da utilizzare, nel nostro caso è stato inserito in “MODEL” e si è scelto di utilizzare “gpt-3.5-turbo”, a cui presto si affida l’api-key per poter utilizzare l’API;
- Prende in input i messaggi da inviare all’API, che devono essere formati in un modo preciso altrimenti non possono essere mandati in esecuzione poiché non conformi al modello;
- Imposta la creatività del modello, che sarà più rigido e preciso con valori di range bassi come [0, 1) e sarà più creativo con valori alti come [1, 2];
- Crea un tunnel di comunicazione tra la macchina (client) e GPT (server).

I messaggi inviati al server devono esser trattati in un certo modo, ed è per questo motivo che viene finalmente definito il concetto di “ruolo”:

Il ruolo è un concetto legato ad ogni messaggio inviato al LLM, ed OpenAI ne mette a disposizione tre tipologie:

- **System**: che tramite delle istruzioni definisce il ruolo dell’AI con cui si sta comunicando;
- **User**: consiste nelle query effettuate dall’utente;
- Assistant: definisce le risposte del modello in base al messaggio con ruolo User.

Per effettuare una richiesta al LLM quindi dobbiamo modellare i messaggi in modo tale da includere in essi, all’inizio del prompt, il ruolo di ognuno di essi.

Una volta definiti i prompt e i ruoli di ognuno di essi possiamo avviare la comunicazione.

Input prompt:

```
SYSTEM_PROMPT = "You're a financial companion and fraud detector \n\
User provides a transaction, and based on your knowledge you have to guess if it is a genuine transaction or a fraudolent transaction \n\
Answer using two sentences explaining the outcome and why you chose it as outcome. You also need to check if the transaction makes sense."


USER_PROMPT= f"\n\
Amount: {transazione['Amount']}\n\
Type: {transazione['Type']}\n\
Old balance of the origin: {transazione['OldBalanceOrig']}\n\
New balance of the origin: {transazione['NewBalanceOrig']}\n\
Old balance of the destination: {transazione['OldBalanceDest']}\n\
New balance of the destination: {transazione['NewBalanceDest']}"

ASSISTANT_PROMPT = '''Use the following format:\n\
Hello, i'm your Financial Assistant and I will assist you to understand this transaction.
Your transaction is likely to be <your outcome>\n\
The reason for the outcome is: <your explanation>'''

messages = build_messages(SYSTEM_PROMPT, USER_PROMPT, ASSISTANT_PROMPT)
```

Capitolo 4.4: Risultati

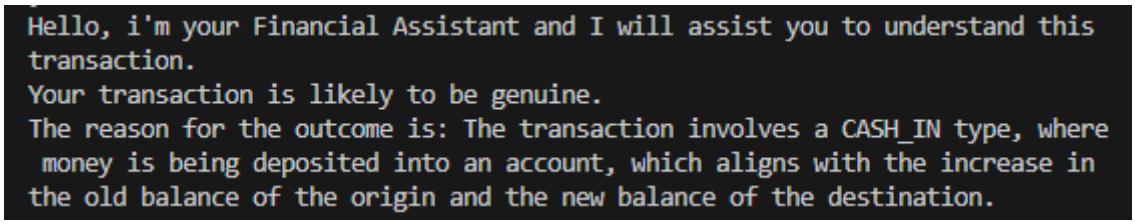

```
  "id": {REDACTED}
  "object": "chat.completion",
  "created": {REDACTED}
  "model": "gpt-3.5-turbo-0125",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "Hello, i'm your Financial Assistant and I will assist you
 to understand this transaction.\nYour transaction is likely to be genuine.\n
The reason for the outcome is: The transaction involves a CASH_IN type, where
 money is being deposited into an account, which aligns with the increase in
the old balance of the origin and the new balance of the destination."
      },
      "logprobs": null,
      "finish_reason": "stop"
    }
  ],
  "usage": {
    "prompt_tokens": 195,
    "completion_tokens": 71,
    "total_tokens": 266
  },
  "system_fingerprint": {REDACTED}
}
```


Output intero del LLM

Si notano caratteristiche importanti - oltre il contenuto - come l'uso dei token, determinati in base alla lunghezza dei prompt in input (prompt tokens) e dell'output (completion tokens), che determinano:

- Il massimo dei token in input: che non può superare 4096;
- Il costo, sottratto all'Usage disponibile, dell'operazione effettuata.

Come possiamo notare, l'output del modello deve essere trattato in modo tale da poter ricevere soltanto il suo contenuto.
Fatto ciò, l'output ottenuto è infatti il seguente:



Output del contenuto della risposta del LLM

Come notiamo, il LLM è capace di effettuare analisi tenendo in conto delle informazioni a lui date; quindi predice, secondo la sua conoscenza, se secondo lui l'operazione ambigua è genuina o fraudolenta notando la coerenza dei dati in ingresso.

Questa è un'operazione che può essere effettuata solamente in questa fase, poiché se la effettuassimo per l'analisi dei dataset interamente sarebbe troppo dispendioso a livello di token; quindi adottiamo separatamente apprendimento supervisionato e LLM.

Capitolo 5: Apprendimento non Supervisionato

L'apprendimento non supervisionato è anch'esso parte dell'apprendimento automatico.
In questo tipo di apprendimento, il modello viene addestrato su un insieme di dati che non presenta etichette. Esistono due tipi di apprendimento non supervisionato:

- **Clustering**: Raggruppa gli elementi dell'insieme in input basandosi sulle somiglianze tra di loro, assegnandogli un'etichetta che identifichi il cluster di appartenenza;
- **Riduzione delle dimensionalità**: Riduce il numero di feature appartenenti agli elementi dell'insieme in input mantenendo le informazioni significative.

È stato utilizzato il modello di clustering **Expectation Maximization** al fine di raggruppare gli elementi del dataset in più clusters imperativamente tramite la probabilità di appartenenza ad un dato cluster.

Il modello utilizzato è stato implementato tramite la libreria sklearn, più precisamente la classe "**Gaussian Mixture Model**", che usa una combinazione di distribuzioni gaussiane per rappresentare i cluster ed ogni punto dati ha una distribuzione di probabilità su tutti i cluster.

L'Expectation Maximization rientra nel calcolo delle probabilità che un dato elemento appartenga ai cluster tramite l'Expectation, mentre poi aggiorna le medie, covarianze e pesi dei parametri per massimizzare le aspettazioni trovate nello step di Expectation.
Questi due passaggi vengono ripetuti fino a che il modello non converge.

Poiché non è necessario, a meno di funzioni future aggiunte, di calcolare la probabilità di un dato nuovo elemento di appartenenza ad un cluster, non sono state usate tutte le funzionalità offerte dal soft clustering, che l'Expectation Maximization permette.

Capitolo 5.1: Motivazioni, insieme di input e insieme di output:

Quest'operazione è effettuata al fine di rilevare tutti quei cluster contenenti un'alta concentrazione di elementi fraudolenti e genuini, dando in ingresso al modello di apprendimento non supervisionato un dataset che contiene le transazioni ma non contiene la loro etichetta.
Questo processo è svolto così che:

- Gli elementi genuini in tali cluster siano ispezionati ulteriormente (**Anomaly detection**, **prioritizzazione delle indagini**);
- Si possano riconoscere le abitudini delle transazioni fraudolente in base ai valori di essi (**Habits detection**).

L'apprendimento viene effettuato sulle seguenti colonne del dataset:
Transazioni[Step, Amount, OldBalanceOrig, NewBalanceOrig, OldBalanceDest, NewBalanceDest]
così da non tener conto del fatto che una transazione sia fraudolenta o meno, lasceremo che l'apprendimento supervisionato faccia le sue supposizioni e inserisca nei cluster gli elementi indipendentemente dalla loro etichetta.

In output riceviamo invece il dataset intero, al cui aggiungiamo il cluster.
Ciò è svolto in modo tale da, nelle analisi, tenere anche conto dell'etichetta oltre che del cluster di ogni elemento.
Questo permetterà le operazioni da svolgere definite nel punto precedente.

L’apprendimento supervisionato potrebbe esser inoltre utilizzato, come funzionalità ulteriore (non implementata), per poter definire i cambi di abitudini delle transazioni fraudolente nel corso del tempo (Step), tramite le seguenti colonne in ingresso:
Transazioni[Step, Amount, IsFraud] tale che tutti gli IsFraud = 1.

Capitolo 5.2: Ottimizzazione dei clusters:

Per poter calcolare il numero di cluster adatto per il modello di GMM è stato utilizzato il Bayesian Information Criterion (BIC), spesso utilizzato con l’algoritmo di EM.

BIC = $-2 * \log L_j\left(\widehat{L_j}\right) * k\log\left(n\right)$

- Lj = Aspettazione del modello;
- $\widehat{L_j}$ = Stimatore di massima verosimiglianza del modello;
- k = numero di componenti del modello;
- n = grandezza dell’insieme in input o numero di osservazioni;
- $klogn$ può esser visto come numero di parametri indipendenti, e serve a penalizzare tutti i modelli che hanno complessità alta;
- $\log L_j\left(\widehat{L_j}\right)$ invece determina quanto il modello si adatti ai dati di apprendimento.

Il BIC cerca quindi modelli che siano abbastanza complessi da adattarsi bene ai dati, ma che non sia troppo complesso da esser poi penalizzato dalla presenza di troppi componenti all’interno del modello.

Un BIC basso definisce il numero di cluster più adatto all’interno del range di componenti che si va a calcolare.

Nel nostro caso otteniamo il seguente grafico dei BIC:

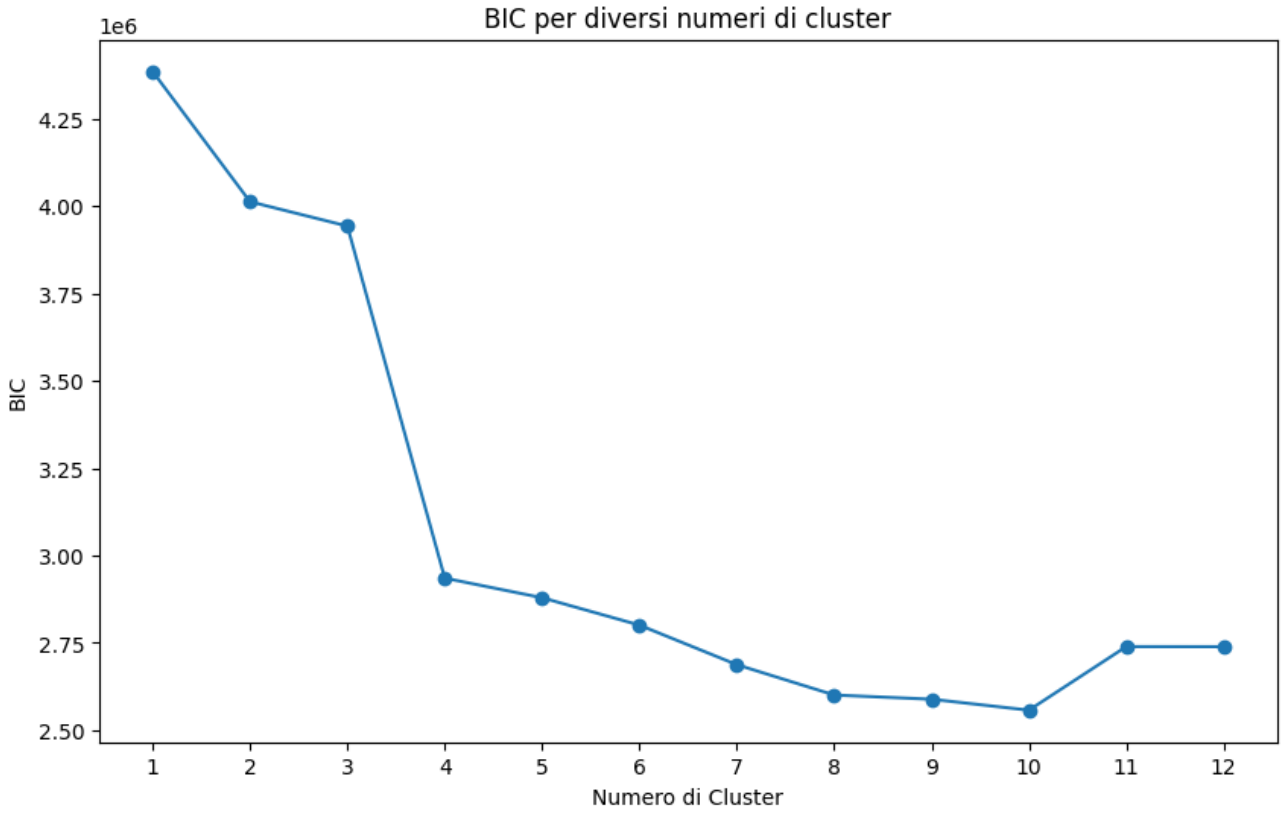


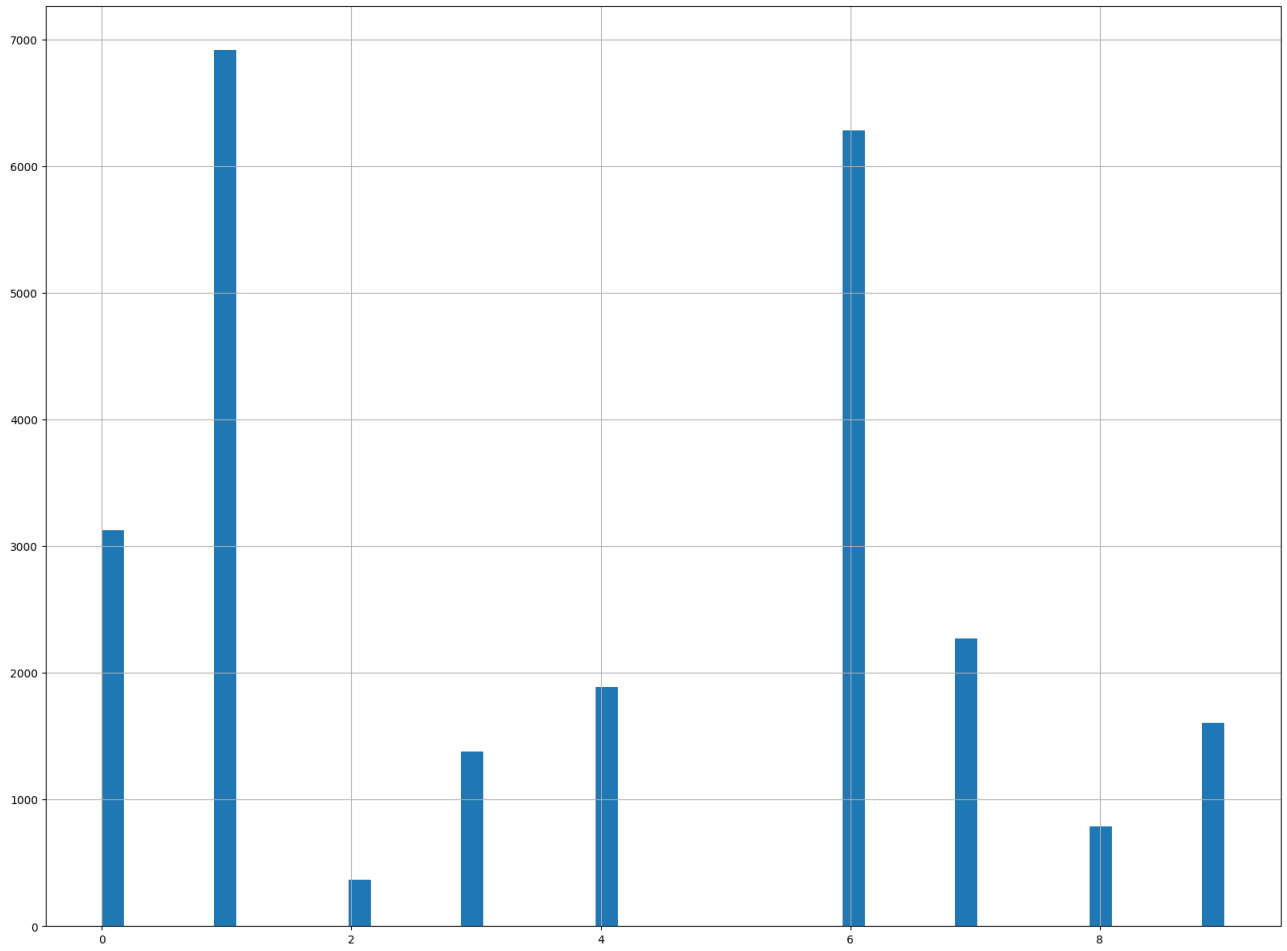
Grafico dei BIC per ogni k cluster

Come possiamo notare, rileviamo una BIC alta nei primi valori data la bassa capacità del modello di adattarsi al dataset, che continua a scendere fino a k=10. Da 10 in poi il BIC comincia ad alzarsi, il che significa che il modello diventa troppo complicato e viene penalizzato dal numero di componenti.

Si procede quindi con numero di cluster k = 10.

Capitolo 5.3: Applicazione del modello Gaussian Mixture Model e Risultati:

Applicando il modello al nostro dataset, notiamo che la distribuzione degli elementi dei cluster non è bilanciata e vede il cluster 5 con solamente 2 elementi.



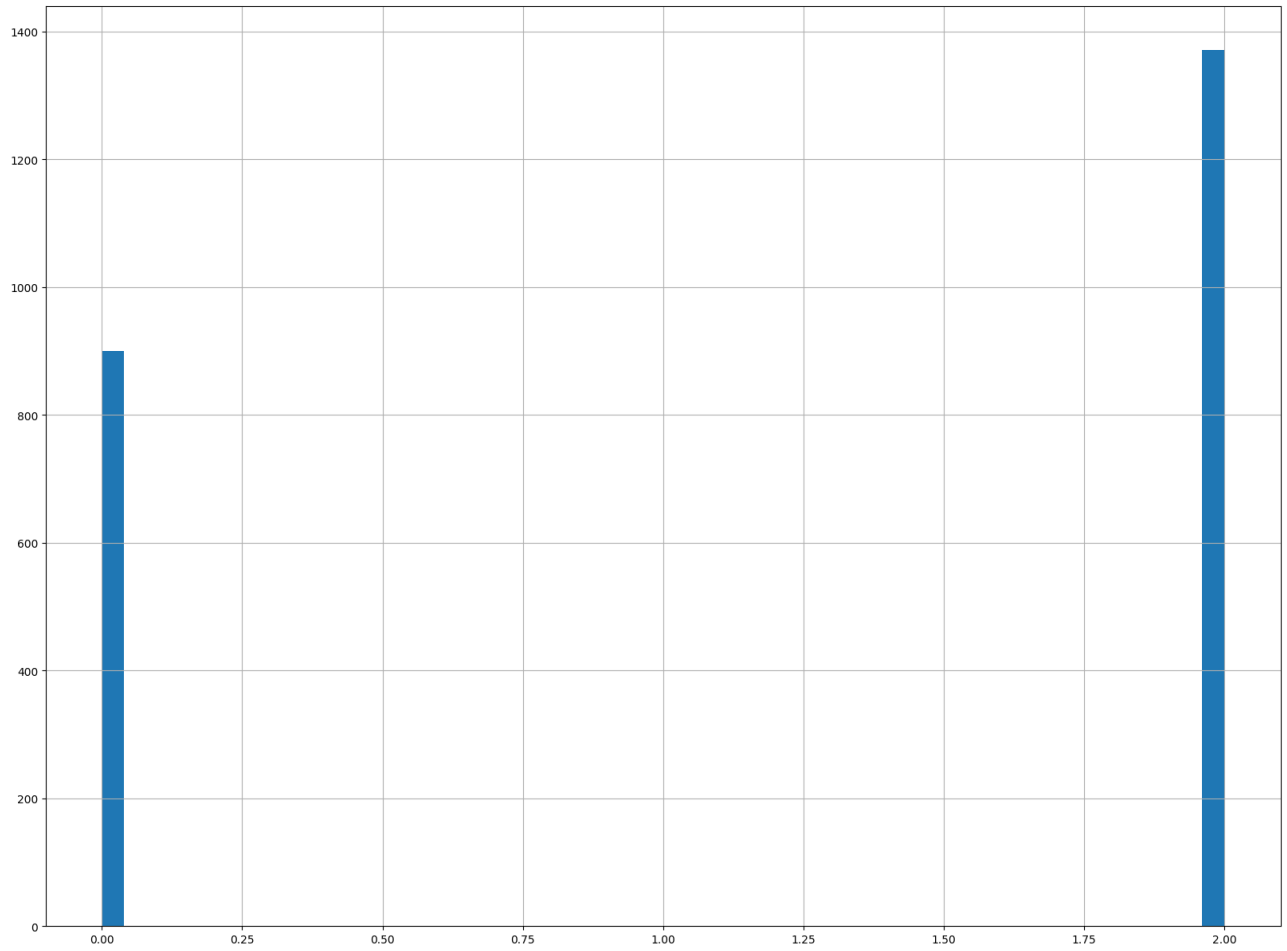
Distribuzione degli elementi nei clusters

Questo potrebbe essere un campanello d'allarme, ma nel nostro caso è normale che i clusters non siano bilanciati poiché i dati sono principalmente di 3 tipi, e se smistati in 10 differenti cluster tendono a raggrupparsi in dataset in base alla loro tipologia, nonostante sia stata esclusa dal calcolo dei clusters.

Il cluster 5 quindi rappresenta elementi molto simili tra loro.

Effettuando un'ulteriore analisi ai valori ottenuti, notiamo la presenza di un cluster con un'alta concentrazione di elementi genuini in cluster all'interno di cui c'è una maggioranza di transazioni fraudolente, il che soddisfa lo scopo del test: la rilevazione di anomalie tra le transazioni genuine e fraudolente.

Di seguito è allegato il cluster che, se si dovesse applicare un'analisi delle transazioni come l'obiettivo suggerirebbe (ma che non verrà applicato poiché non ne ho le conoscenze in contesto finanziario), dovrebbe essere messo in testing:



Cluster contenente elementi genuini "rischiosi"

Come notiamo, circa 900 elementi genuini sono presenti in un cluster contenente circa 1400 elementi fraudolenti.

Capitolo 6: Ragionamento logico tramite Prolog

Capitolo 6.1: Introduzione al Ragionamento Logico

Ai fini di poter stabilire relazioni tra gli utenti, le transazioni, e le regole che si possono creare tra esse e le query ad esse applicabili.

Viene costruita quindi una Knowledge Base partendo dal dataset preprocessato, a cui aggiungiamo i risultati del clustering.

Una Knowledge Base è un insieme di proposizioni che sono specificate come vere, e quindi assiomi.
Nel nostro caso la Knowledge Base è composta da atomi, ovvero proposizioni, primitivi ed atomi derivati:

- Atomi primitivi: proposizioni definite attraverso i **fatti**;

- Atomi derivati: proposizioni definite attraverso **regole**.

La Knowledge Base sarà formata da:

- Fatti: realtà immutabili che contengono informazioni riguardo il concetto rappresentato;
- Regole: dichiarazioni che definiscono se una proposizione è vera in base ai fatti prestabiliti e alle osservazioni effettuabili basandosi su tali fatti. Le regole derivano tramite inferenza delle relazioni logiche stabilite tra le proposizioni della knowledge base.

Prolog è un linguaggio di programmazione dichiarativo di ragionamento logico, che prevede la presenza di query: interrogazioni logiche che si possono effettuare rivolgendosi alla KB, tramite linguaggio di programmazione logica.

La programmazione logica è una branca del Predicate Calculus, all'interno di cui gli atomi di cui è composta la Knowledge Base presentano una struttura interna comprendente di costanti e variabili logiche.
Tali atomi poi formano il mondo chiuso su cui stiamo lavorando in base alle loro proprietà individuali e alle relazioni che possono avere tra più individui.

Capitolo 6.2: Implementazione della Knowledge Base

All'interno della Knowledge Base del sistema sono contenuti:

Fatti:

- utente: contenente le informazioni riguardo l'utente, definito dalla sua carta e il suo patrimonio su di essa;
- transazione: contenente le informazioni riguardo le transazioni, un subset delle transazioni del dataset, definite da: utente mittente, utente destinatario, tipo, ammonto, Cluster, IsFraud;

Regole:

- soci(X, Y): Definisce che se esiste una transazione tra un utente1 mittente e un utente2 destinatario, ed esiste una transazione tra utente2 mittente e utente1 destinatario, allora i due utenti sono soci;

```
soci(X, Y):- transazione(_, X, Y, _, _, _, _, _),
transazione(_, Y, X, _, _, _, _, _).
```

- lista_soci(X, ListaSoci): Definisce la lista di tutti i soci di un dato utente nella Knowledge Base;

```
lista_soci(X, ListaSoci) :-
    findall(Y, cluster_soci(X, Y), ListaSoci).
```

- cluster_fraudolento(): Definisce un cluster che contiene più transazioni fraudolente che transazioni ambigue o genuine;

```
cluster_fraudolento(Cluster) :-
    findall(ID, transazione(ID, _, _, _, _, _, Cluster, 0),
Genuine),
    findall(ID, transazione(ID, _, _, _, _, _, Cluster, 1),
Ambigue),
    findall(ID, transazione(ID, _, _, _, _, _, Cluster, 2),
Fraudolente),
    length(Genuine, NumGenuine),
    length(Ambigue, NumAmbigue),
    length(Fraudolente, NumFraudolente),
    NumFraudolente > NumAmbigue,
    NumFraudolente > NumGenuine.
```

- lista_cluster_fraudolenti(ListaCluster): Definisce la lista dei cluster che vedono più transazioni fraudolente che ambigue o genuine;

```
lista_cluster_fraudolenti(ListaCluster) :-
    findall(Cluster, cluster_fraudolento(Cluster),
ListaCluster).
```

- utenti_piu_ricchi(Top, N): Definisce la lista di utenti più ricchi ed è, manipolabile tramite query, per mostrare i primi N elementi dove N può variare e deve essere maggiore di 0;

```
utenti_piu_ricchi(Top, N) :-
    N > 0,
    findall(Amount-User, utente(User, Amount), Users),
    keysort(Users, SortedUsers),
    reverse(SortedUsers, ReverseSortedUsers),
    primi(N, ReverseSortedUsers, Top).
```

- primi(N, List, Result): regola ausiliaria necessaria perché altrimenti era impossibile prendere i primi N elementi dalla lista dopo averli ordinati per value.

```
primi(N, List, Result) :-
    length(Result, N),
    append(Result, _, List).
```

Tali regole, se combinate con delle query, permettono di visitare le funzioni offerte all'azienda.
La lista di funzioni è espandibile e potrebbe offrirne di molte altre, come l'analisi delle abitudini degli utenti e le loro transazioni.

Le query che potrebbero essere applicate sono del tipo:

?- lista_cluster_fraudolenti(Lista)
?- utenti_piu_ricchi(Top, 100)
?- soci(X, Y)
?- lista_soci(c1375978020, Lista)

Sviluppi Futuri

Nonostante le funzionalità offerte dal sistema siano adeguate ai suoi scopi, è presente una mancanza nel numero delle funzionalità offerte tramite programmazione logica, che è per il momento scarna avente solamente 4query possibili.

Alcune predizioni potrebbero essere inoltre migliorate effettuando un'operazione di feature selection avviando un algoritmo non supervisionato prima dell'apprendimento supervisionato, cercando di definire se effettivamente le colonne utilizzate siano tutte necessarie.
Tale operazione non è stata effettuata per rimanere il più veritieri possibili al dataset, ma potrebbe esser svolta in sviluppi futuri se ritenuta necessaria.

Si potrebbe, al contrario, trovare un modo per includere i nomi degli utenti nel modello supervisionato senza provocare troppa varianza e quindi overfitting ai dati di addestramento.
Ciò si potrebbe attuare dividendo di una costante il valore delle due colonne, così da far sì che abbiano un'importanza ma non estrema.

Riferimenti bibliografici:

[1] D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. 3rd ed. Cambridge University Press.
[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019
[3] Introduzione ai modelli linguistici di grandi dimensioni (LLM) | Machine Learning | Google for Developers
[4] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, A. Mian: A Comprehensive Overview of Large Language Models
[5] Adam J. Stewart, Zaid Qureshi: Prolog Documentation
[6] Sickit-Learn documentation