CREDIT CARD FRAUD DETECTION

Documentazione Caso di Studio Ingegneria della Conoscenza A.A. 2022/23

Gruppo di lavoro:

- Dafne Spaccavento, d.spaccavento@studenti.uniba.it, MATRICOLA: 737615
- Gaetano Schiralli, g.schiralli5@studenti.uniba.it, MATRICOLA: 746460

Repository GitHub:

https://github.com/Nanuccio01/CREDIT_CARD_FRAUD_DETECTION/tree/main

Sommario

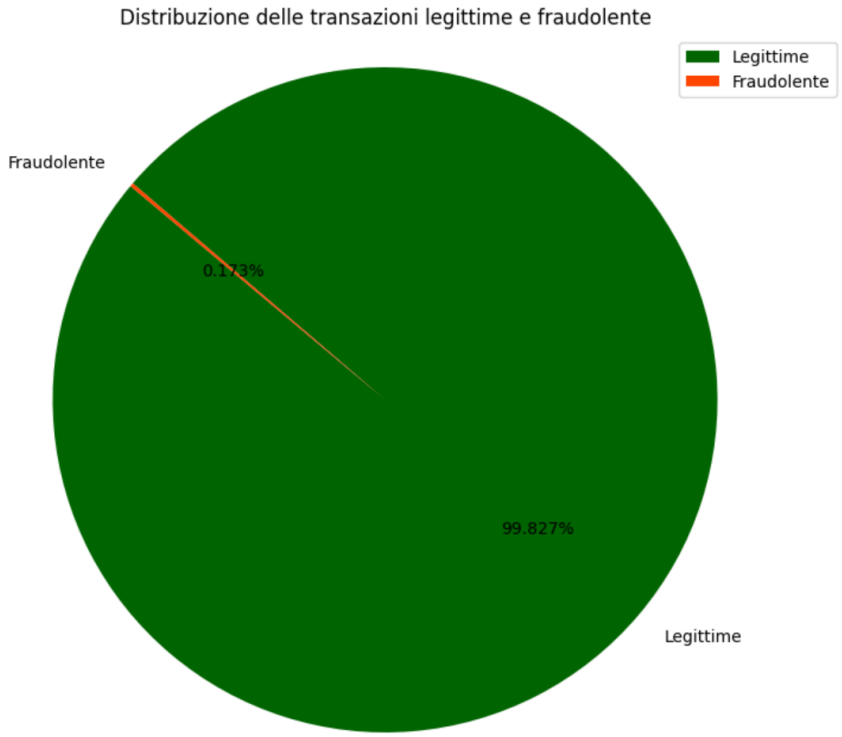
1. Definizione dell'Obiettivo e Comprensione dei Dati

• **L'obiettivo:**

L'obiettivo di questo progetto è sviluppare un modello in grado di individuare le transazioni fraudolente all'interno di un dataset contenente transazioni effettuate con carte di credito. Il dataset in questione contiene transazioni effettuate da titolari di carte di credito europee nel mese di settembre 2013.

• **Analisi Esplorativa dei Dati:**

Il dataset è composto da transazioni che si sono verificate in due giorni, con un totale di 284.807 transazioni. All'interno di queste transazioni, sono presenti 492 casi di frode. Le variabili di input sono tutte numeriche. Il dataset presenta uno sbilanciamento significativo tra le classi, poiché la classe positiva (frodi) costituisce lo 0.172% di tutte le transazioni.



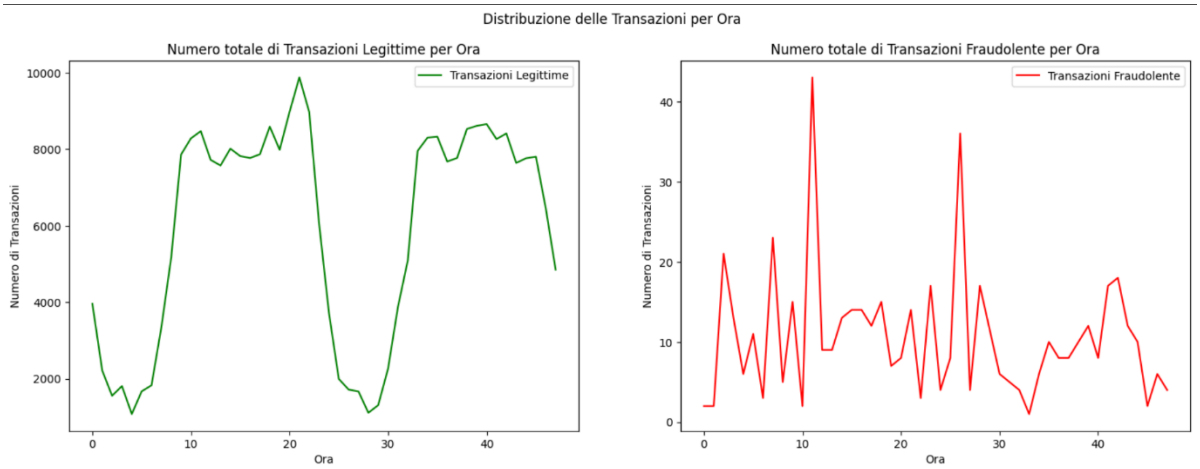
- Le feature V1, V2, ..., V28 sono campi float e rappresentano le principali componenti ottenute come risultato di una trasformazione PCA (Principal Component Analysis), perché a causa delle tutele privacy, il nome di questi campi è reso anonimo. Per implementare una trasformazione PCA, le feature devono essere preventivamente scalate. In questo caso, quindi, tutte le feature V1, V2, ..., V28 sono state scalate o almeno è ciò che assumiamo abbiano fatto gli sviluppatori del dataset.

- Le uniche feature che non sono state trasformate con PCA sono 'Time' e 'Amount', entrambe intere. La feature 'Time' rappresenta i secondi trascorsi tra ogni transazione e la prima transazione nel dataset, mentre la feature 'Amount' rappresenta l'importo della transazione. La feature 'Class' è la variabile di risposta e assume il valore 1 in caso di frode e 0 altrimenti.

Successivamente abbiamo scelto di analizzare e descrivere approfonditamente, solo le colonne 'Time', 'Amount' e 'Class' (tralasciando le feature V1, V2, ..., V28), per ottenere una visione dettagliata di alcune delle caratteristiche chiave dei dati che possono avere un impatto significativo sull'analisi delle frodi su carte di credito:

- "Amount" (Importo della Transazione): Si analizza questa colonna per capire la distribuzione degli importi ed il totale delle transazioni nel dataset. L'importo delle transazioni potrebbe variare ampiamente e potrebbe essere utile comprendere se ci sono trend o pattern specifici nella distribuzione degli importi per le transazioni legittime o fraudolente.

- "Time" (Tempo della Transazione): Si esamina questa colonna per identificare qualsiasi modello temporale nei dati. Potrebbe esserci una correlazione tra i momenti delle transazioni e la probabilità di frode. Analizzando questa colonna, si potrebbero individuare intervalli di tempo in cui si verificano più frodi o altre tendenze temporali interessanti, in quanto nel dataset sono presenti più transazioni per ogni secondo analizzato.



Da questo grafico possiamo notare che le transazioni fraudolente hanno una distribuzione più uniforme nel tempo rispetto alle transazioni legittime: esse sono distribuite in modo più uniforme nel tempo, al contrario delle transazioni legittime che subiscono un calo durante le ore notturne del fuso orario europeo.

- "Class" (Classe di Transazione - Legittima(0) o Fraudolenta(1)): Si inserisce questa colonna perché rappresenta la variabile target dell'analisi delle frodi. Sebbene la si inserisca solo per capire la distribuzione delle classi nel dataset (già analizzate in precedenza), questo campo è nuovamente analizzato per completezza.

Qui troviamo riportate le statistiche descrittive delle tre features sopra descritte:

Statistiche descrittive

	count	mean	std	min	25%	50%	75%	max
Amount	284807.00	88.35	250.12	0.00	5.60	22.00	77.16	25691.16
Time	284807.00	94813.86	47488.15	0.00	54201.50	84692.00	139320.50	172792.00
Class	284807.00	0.00	0.04	0.00	0.00	0.00	0.00	1.00

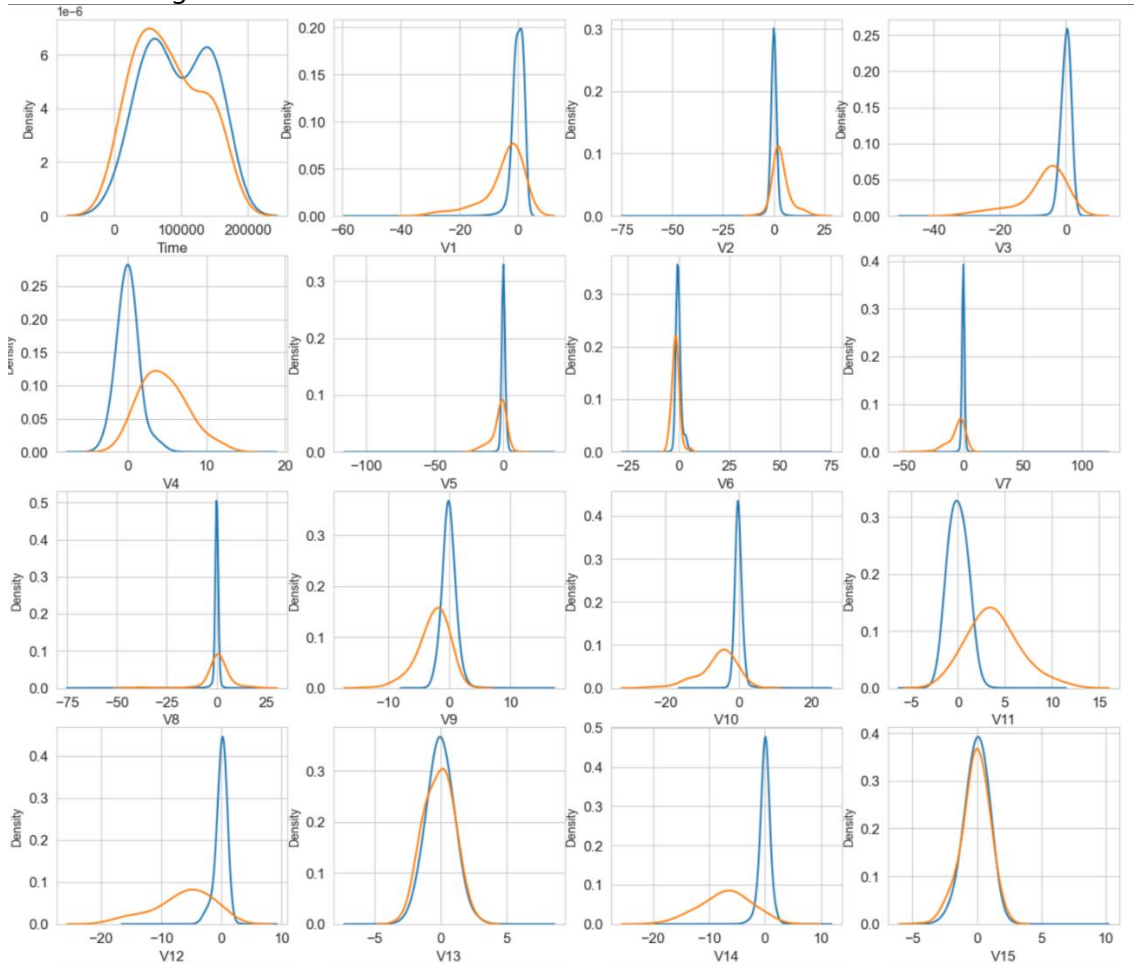
Guardando alla caratteristica "Time", possiamo confermare che i dati contengono 284807 transazioni, distribuite durante 2 giorni consecutivi (o 172792 secondi).

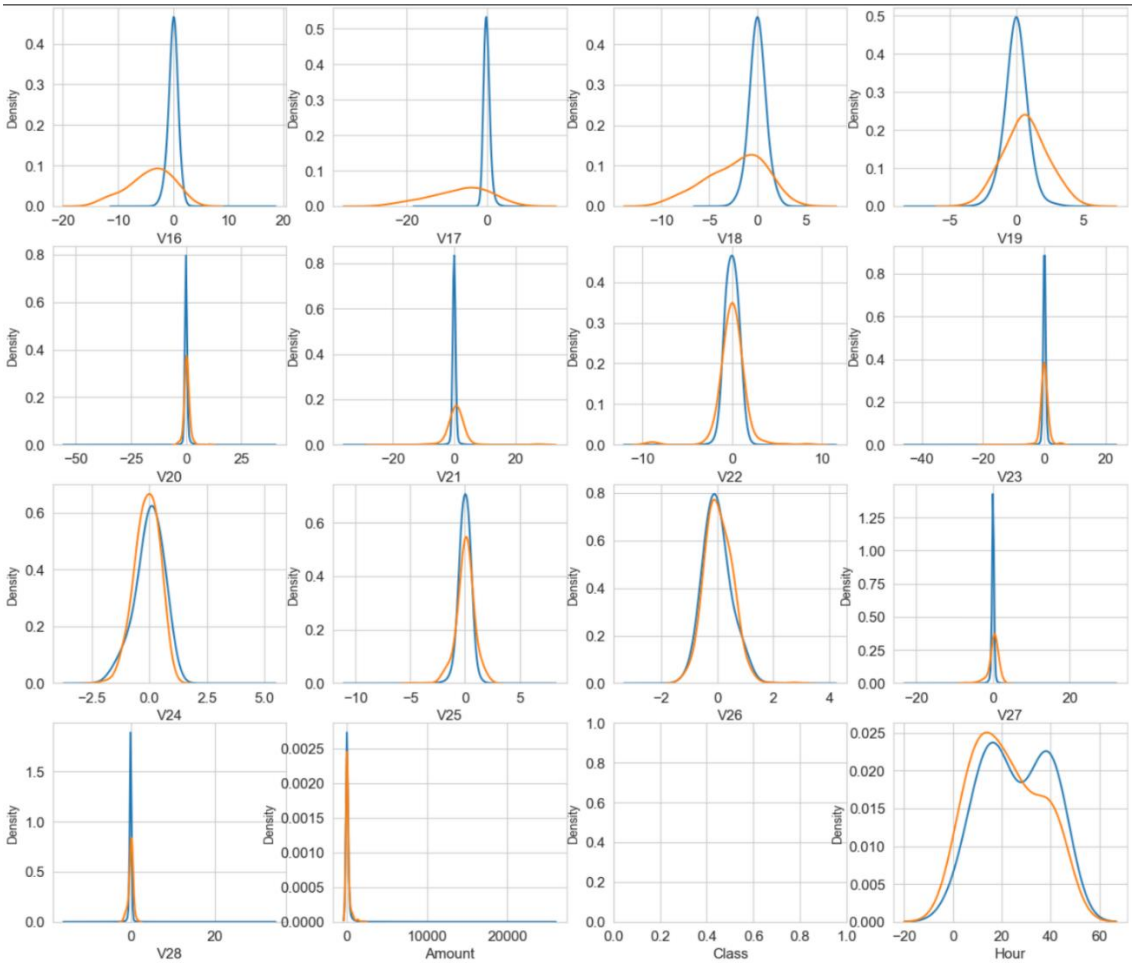
Dopo aver esplorato inizialmente i dati, possiamo affermare a colpo d'occhio di non aver notato nessuna ripetizione o comportamento significativo tra essi.

Per avere un quadro più generale delle correlazioni, quindi, si sono utilizzati i grafici KDE che mostrano la stima delle densità delle caratteristiche per due classi: Classe 0 (transazioni legittime) e Classe 1 (transazioni fraudolente):

- I grafici servono a visualizzare come le distribuzioni delle caratteristiche differiscano tra le due classi; infatti, consentono di vedere la forma approssimativa delle distribuzioni delle caratteristiche per entrambe le classi. Se ci fossero differenze significative nelle distribuzioni tra le classi per una particolare caratteristica, potrebbe indicare che essa è utile per distinguere tra

transazioni legittime e fraudolente.





Per alcune delle caratteristiche possiamo osservare una buona selettività in termini di distribuzione per i due valori della classe:

- V4 e V11 hanno distribuzioni chiaramente separate per i valori di Classe 0 e 1.
- V12 e V14 sono parzialmente separati.
- V1, V2, V3 e V10 hanno un profilo piuttosto distinto.
- V25, V26 e V28 hanno profili simili per i due valori della Classe.

In generale, con poche eccezioni (Tempo e Importo), la distribuzione delle caratteristiche per le transazioni legittime (valori di Classe = 0) è centrata intorno a 0, talvolta con una lunga coda su uno dei lati estremi. Allo stesso tempo, le transazioni fraudolente (valori di Classe = 1) hanno una distribuzione asimmetrica (sbilanciata).

Basandoci sui grafici KDE, potremmo fare delle scelte informate su quali feature includere nel modello successivamente. Se alcune caratteristiche avessero distribuzioni molto simili tra le due classi, avremmo potuto decidere di escluderle dal modello, riducendo così la complessità e migliorando le prestazioni.

2. Preparazione dei Dati

Gestione dei dati mancanti e non utilizzabili: Una volta caricato il dataset completo si analizza la sua struttura. Si controlla la presenza di valori mancanti nel Dataset. In questo caso nessun problema vien creato, in quanto tutte le colonne presentano dei valori.

	Time	V16	Amount	V28	V27	V26	V25	V24	V23	V22	...	V10	V9	V8	V7	V6	V5	V4	V3	V2	Class
Totale	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Percentuale	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Successivamente analizzando il dataset per la ricerca di valori nulli come importo, si è notato che:

- Il numero delle transazioni fraudolente con importo nullo (Amount=0) è: 27
- Il numero delle transazioni legittime con importo nullo (Amount=0) è: 1798

Inizialmente si stava considerando l'eliminazione di questi campi poiché non sembrava esserci un motivo o uno scopo chiaro per la loro presenza, in quanto una transazione non si potrebbe definire valida con un ammontare pari a zero. Ricercando ulteriormente però, si è appurato che esistono sia venditori che effettuano una transazione di prova per verificare gli estremi della carta bancaria, e sia siti web di lotterie o concorsi dove il vincitore effettua una transazione con ammontare pari a zero. Considerando codeste variabili reali quindi, tali righe sono state accettate come valide nel dataset.

Inoltre, in questa fase, andremo a scalare le colonne Time e Amount, per avere valori simili alle altre colonne precedentemente scalate in seguito alla trasformazione PCA.

	scaled_amount	scaled_time	V1	V2	V3	V4	V5	V6	V7	V8	...
0	1.783274	-0.994983	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	...
1	-0.269825	-0.994983	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	...
2	4.983721	-0.994972	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	...
3	1.418291	-0.994972	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	...
4	0.670579	-0.994960	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	...

3. Overfitting

• Concetto e Tecniche utilizzate per mitigare l'Overfitting:

Uno dei problemi che potrebbe creare lo sbilanciamento del nostro dataset originale è quello dell'overfitting portando, dunque, il modello di apprendimento automatico a adattarsi troppo ai dati di addestramento. Di conseguenza, il modello potrebbe avere una performance eccezionalmente buona sui dati di addestramento, ma si comporterebbe in maniera pessima su nuovi dati che non ha mai elaborato prima.

Dunque, se utilizzassimo questo dataframe come base per i nostri modelli predittivi e per le analisi, potremmo ottenere errori, in quanto gli algoritmi probabilmente "assumeranno" che la maggior parte delle transazioni non siano truffe (ricordiamo che sono solo lo 0.172% di tutte le transazioni).

Per questo andremo ad utilizzare e a mettere a confronto le seguenti tecniche:

- Creare un bilanciamento delle classi del dataset (sub-Sampling): In questo scenario, il nostro subsample sarà un dataframe con un rapporto 50/50 tra transazioni fraudolente e non fraudolente.

- Utilizzare la Cross-validation stratificata: Abbiamo utilizzato la 5-fold stratified cross validation per garantire che ogni fold della cross validation mantenesse la stessa proporzione tra le classi "legittimo" e "fraudolento" presente nel dataset completo. Questo per contribuire ad evitare che

il modello apprendesse troppo dalle transazioni "legittime" e garantiva una valutazione accurata delle prestazioni su entrambe le classi.

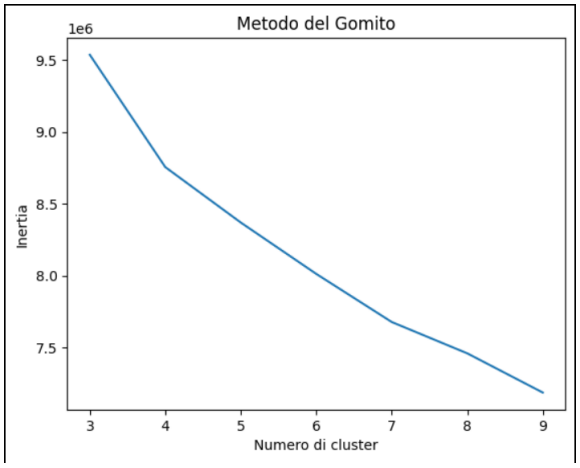
4. Apprendimento Non Supervisionato e Clustering

• **Applicazione del Metodo di Clustering: K-Means**

In questa fase abbiamo scelto di impiegare l'algoritmo di K-Means per condurre un'analisi di clustering sui dati. L'obiettivo principale di questa fase è identificare possibili sottoclassi o gruppi di transazioni all'interno delle due classi principali, "legittime" e "fraudolente".

Tuttavia, come abbiamo già evidenziato, è importante notare che è già presente una divisione in due classi ben definita e utilizzata come etichetta di apprendimento supervisionato nei nostri modelli. Pertanto, un approccio di K-Means con K=2 (due cluster) non sarebbe appropriato, poiché verosimilmente produrrebbe risultati simili alla suddivisione preesistente tra "legittime" e "fraudolente". Questo non aggiungerebbe valore aggiunto alla nostra analisi.

Invece, intendiamo utilizzare K-Means con un valore di K superiore a 2, consentendo così all'algoritmo di identificare gruppi o cluster all'interno delle due classi principali. Per trovare il k ottimale andremo ad utilizzare il metodo del gomito, provando diversi valori tra 3 e 10.

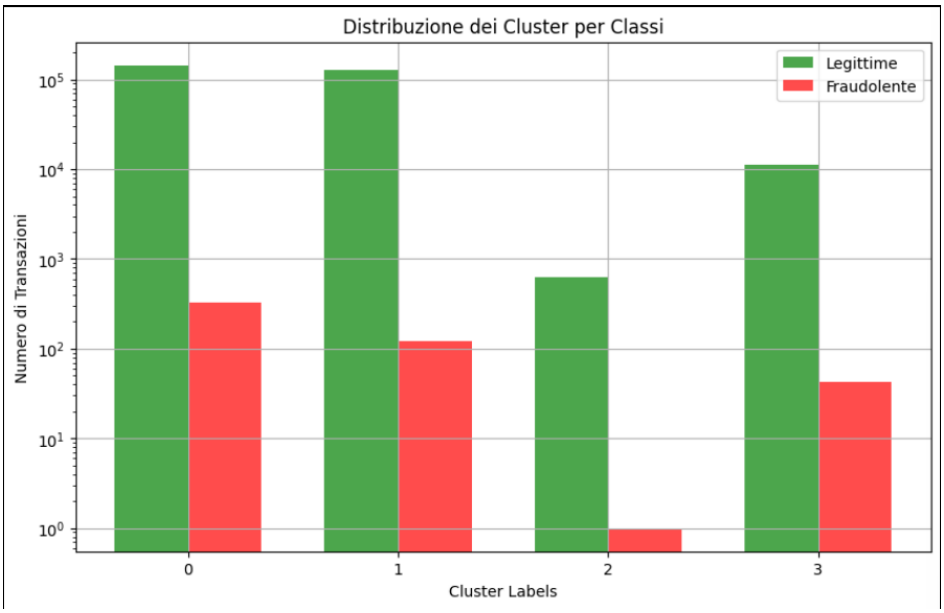


Nel grafico si evince che sia 4 che 7 potrebbero rappresentare il "gomito" della curva.

• **Interpretazione dei cluster ottenuti**

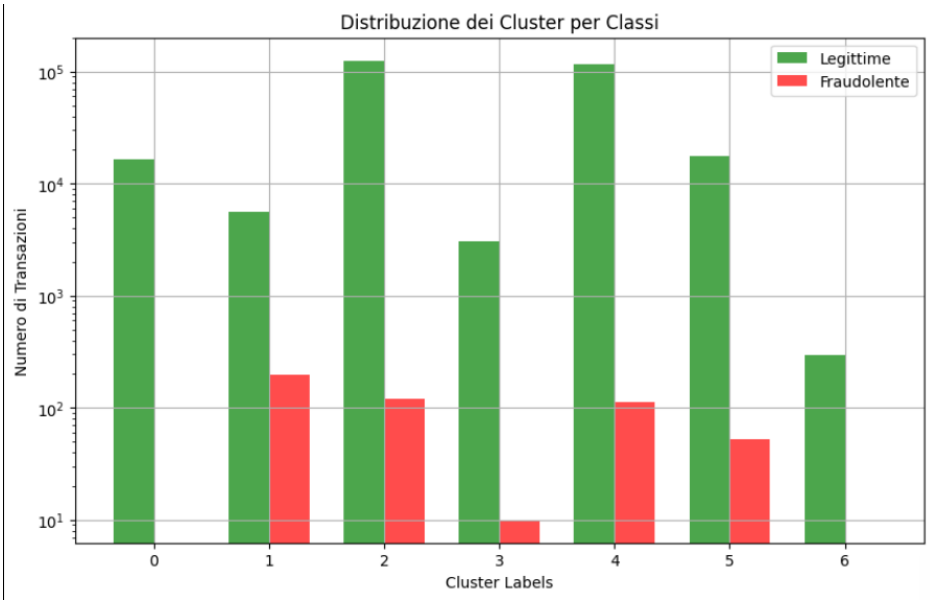
Iniziamo analizzando i risultati utilizzando k=4: Per valutare come le quattro label dei cluster fossero distribuite tra le transazioni legittime e fraudolente, abbiamo calcolato il rapporto tra il numero di transazioni associate a ciascuna label di cluster e il totale delle transazioni legittime e fraudolente nel dataset. Questo rapporto ha fornito una panoramica chiara della distribuzione delle label dei cluster all'interno di entrambe le classi. Tale analisi è risultata inconcludente e non abbiamo identificato eventuali pattern distinti all'interno di ciascuna classe. I rapporti sono pressoché simili, tranne per la label "1" che ha una percentuale più alta nella classe delle transazioni legittime.

Class	0	1	Legit Ratio	Fraud Ratio
cluster_labels				
0	144213	328	0.507230	0.666667
1	128219	120	0.450975	0.243902
2	628	1	0.002209	0.002033
3	11255	43	0.039586	0.087398



Procediamo con l'analisi dei risultati utilizzando k=7: Abbiamo dunque ripetuto il medesimo procedimento di analisi delle label dei cluster, ma questa volta abbiamo impostato il valore di k a 7. Questo ci ha permesso di suddividere ulteriormente le transazioni in sottogruppi più specifici. Durante questa analisi, abbiamo notato un pattern significativo nella label 1 del clustering, in quanto si è evidenziata un'alta percentuale di transazioni fraudolente, approssimativamente intorno al 40%. Tale distribuzione è in netto contrasto con la classe delle transazioni legittime, in cui la percentuale di distribuzione per la stessa label è stata di circa il 2%. Tuttavia, è importante sottolineare che nelle altre label dei cluster non sono emerse differenze significative tra le due classi, il che suggerisce che il clustering ha identificato un pattern distintivo principalmente nella label 1, mentre le altre label sembrano essere distribuite in modo simile tra le transazioni legittime e fraudolente.

Class	0	1	Legit Ratio	Fraud Ratio
cluster_labels				
0	16440	0	0.057823	0.000000
1	5600	198	0.019696	0.402439
2	125252	119	0.440540	0.241870
3	3030	10	0.010657	0.020325
4	116019	113	0.408065	0.229675
5	17678	52	0.062178	0.105691
6	296	0	0.001041	0.000000



Secondo la nostra valutazione, per quanto con k=7 si sia evidenziata una differenza evidente nella classe di transazioni fraudolente con label 1, non riteniamo che sia utile inserire le etichette evidenziate nel clustering nella successiva fase di apprendimento supervisionato.

5. Classificazione Transazioni

Addestramento modelli sull'intero Dataset

• Introduzione

In questa analisi si prende in considerazione l'intero Dataset. Nelle transazioni, le features di input saranno le colonne numeriche, "Time", "Amount" e le features da "V1" a "V28". La feature target sarà la colonna "Class", che indica se una transazione è fraudolenta o legittima.

Gli "esempi di training" saranno un sottoinsieme del dataset con tutte le colonne, inclusa la colonna "Class". Questi esempi verranno utilizzati per addestrare il modello.

Gli "esempi di test" saranno un altro sottoinsieme del dataset contenente solo le colonne numeriche (senza la colonna "Class"). Questi esempi verranno utilizzati per testare il modello addestrato e fare previsioni sulle classi delle transazioni.

• Scelta degli iperparametri

Per selezionare gli iperparametri ottimali per i nostri modelli di apprendimento abbiamo utilizzato il metodo di ricerca della griglia, noto come "Grid Search". Questo è una tecnica di ricerca esaustiva che ci ha permesso di esplorare un insieme predefinito di combinazioni di iperparametri e determinare quelle che conducono alle prestazioni migliori per i nostri modelli.

Esempi di iperparametri trovati con GridSearch per i diversi classificatori addestrati su tutto il Dataset:

```
KNeighborsClassifier
Migliori iperparametri: {'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'distance'}
Miglior score: 0.940272514714182
DecisionTreeClassifier
Migliori iperparametri: {'max_depth': 5, 'min_samples_split': 5}
Miglior score: 0.9224945577682819
RandomForestClassifier
Migliori iperparametri: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimat
ors': 300}
Miglior score: 0.9491493993388695
GradientBoostingClassifier
Migliori iperparametri: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200}
Miglior score: 0.9402644521486737
AdaBoostClassifier
Migliori iperparametri: {'algorithm': 'SAMME.R', 'learning_rate': 0.5, 'n_estimators': 200}
Miglior score: 0.9478755139885513
LogisticRegression
Migliori iperparametri: {'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}
Miglior score: 0.9440457953720873
```

Esempi di iperparametri trovati con GridSearch per i diversi classificatori addestrati sul subsample:

```
KNeighborsClassifier
Migliori iperparametri: {'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'uniform'}
Miglior score: 0.940272514714182
DecisionTreeClassifier
Migliori iperparametri: {'max_depth': 3, 'min_samples_split': 2}
Miglior score: 0.9275900991695558
RandomForestClassifier
Migliori iperparametri: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimat
ors': 300}
Miglior score: 0.9491493993388695
GradientBoostingClassifier
Migliori iperparametri: {'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 100}
Miglior score: 0.9415383374989921
AdaBoostClassifier
Migliori iperparametri: {'algorithm': 'SAMME.R', 'base_estimator': None, 'learning_rate': 0.5, 'n_estimators': 200}
Miglior score: 0.9478755139885513
LogisticRegression
Migliori iperparametri: {'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}
Miglior score: 0.9440457953720873
```

Non si sono ricercati gli iperparametri del Naive Bayes in quanto, esso è noto per la sua semplicità ed è generalmente meno sensibile agli iperparametri rispetto ad altri algoritmi di machine learning.

Non si sono ricercati gli iperparametri di tutti i nostri modelli di apprendimento durante la 5-fold Stratified Cross Validation a causa di un altissimo costo computazionale.

Si sono considerati più parametri durante la ricerca nella param_grid di ogni singolo modello addestrato sul Subsample, in quanto la ricerca è risultata molto più veloce in termini di tempo e meno complessa.

Gli score ed i valori che andremo ad analizzare successivamente corrisponderanno ai modelli addestrati con gli iperparametri trovati, in quanto abbiamo notato un miglioramento significativo rispetto ai modelli addestrati con i parametri di default. Questo miglioramento si osserva in maniera più evidente per gli alberi di decisione.

- **KNN**

Considerando che il nostro dataset contiene una porzione significativa di dati privati i quali non possono essere selezionati o esclusi a causa della mancanza di conoscenza sul loro contenuto, potrebbe essere prudente utilizzare tutte le features disponibili nel contesto dell'algoritmo k-Nearest Neighbors (k-NN). Questo approccio ci permette di sfruttare tutte le informazioni disponibili per effettuare le previsioni, tenendo presente che alcune features potrebbero contenere informazioni utili.

Tuttavia, siamo consapevoli dei rischi associati all'utilizzo di tutte le features, specialmente quando il dataset contiene dati sensibili o potenzialmente rumorosi. L'aumento delle dimensioni dello spazio delle

feature potrebbe comportare distorsioni nelle misure delle distanze tra le osservazioni, influenzando le prestazioni del modello k-NN.

```
KNeighborsClassifier Classification_report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.95      0.74      0.83        98

    accuracy                           1.00     56962
   macro avg       0.97      0.87      0.92     56962
weighted avg       1.00      1.00      1.00     56962
```

	Metrica	Valore
0	Precision	0.948052
1	Recall	0.744898
2	F1 Score	0.834286
3	Accuratezza Test	0.999491
4	Accuratezza Train	1.000000

I risultati infatti rispecchiano quanto appena detto, offrendo un valore medio di Recall per l'individuazione delle vere transazioni fraudolente.

Gli iperparametri utilizzati sono: {'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'distance'}. Nel caso della metrica 'manhattan', viene utilizzata la distanza di Manhattan (anche chiamata distanza di città) per calcolare la distanza tra i punti, distanza che è usata anche per la pesatura 'weights' tra le istanze vicine. 'n_neighbors': 5: invece, è il numero di vicini più vicini da considerare quando si effettuano le previsioni per un'istanza.

• **Alberi di decisione:**

- **DecisionTreeClassifier**: Abbiamo iniziato questo algoritmo che costruisce un singolo albero decisionale con {'max_depth': 5, 'min_samples_split': 5}. Per 'max_depth' intendiamo la massima profondità dell'albero decisionale e per 'min_samples_split' il numero minimo di campioni richiesti in un nodo.

  Questo ci ha permesso di effettuare un primo passo nell'esplorazione delle potenzialità di base degli alberi decisionali.

```
DecisionTreeClassifier Classification_report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.91      0.80      0.85        98

    accuracy                           1.00     56962
   macro avg       0.95      0.90      0.92     56962
weighted avg       1.00      1.00      1.00     56962
```

	Metrica	Valore
0	Precision	0.906977
1	Recall	0.795918
2	F1 Score	0.847826
3	Accuratezza Test	0.999508
4	Accuratezza Train	0.999583

Ecco alcuni dei principali algoritmi ensemble che abbiamo utilizzato:

- **RandomForest:** Successivamente, abbiamo implementato il RandomForest che combina diversi alberi decisionali creati su sottoinsiemi casuali dei dati e delle feature.
  Questo ha migliorato gli score dei modelli, ottenuti fino ad ora, aumentando però la complessità ed i tempi di esecuzione.

  Gli iperparametri utilizzati sono: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 300}. Tralasciando quelli precedentemente descritti per 'max_features' intendiamo il numero massimo di caratteristiche da considerare quando si cerca il miglior split in ciascun albero. Con 'sqrt', verrà utilizzata la radice quadrata del numero totale di caratteristiche.

```
RandomForestClassifier Classification_report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.97      0.74      0.84        98

    accuracy                           1.00     56962
   macro avg       0.99      0.87      0.92     56962
weighted avg       1.00      1.00      1.00     56962
```

	Metrica	Valore
0	Precision	0.973333
1	Recall	0.744898
2	F1 Score	0.843931
3	Accuratezza Test	0.999526
4	Accuratezza Train	0.999675

- **AdaBoost:** è un algoritmo che assegna maggior peso agli esempi classificati erroneamente, permettendo agli alberi successivi di concentrarsi sulle aree difficili da classificare.

  Gli iperparametri utilizzati sono {'algorithm': 'SAMME.R', 'learning_rate': 0.5, 'n_estimators': 200}. L'algoritmo SAMME.R è specificamente adatto per la classificazione binaria, mentre il tasso di apprendimento (Un valore più alto come 0.5 significa un apprendimento più aggressivo, mentre un valore più basso avrà un apprendimento più lento) e il numero di stimatori regolano il processo di addestramento e la complessità dell'ensemble risultante.

  I risultati ottenuti non sono migliori e si classificano in linea con quelli ottenuti precedentemente da altri modelli.

```
AdaBoostClassifier Classification_report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.95      0.72      0.82        98

    accuracy                           1.00     56962
   macro avg       0.97      0.86      0.91     56962
weighted avg       1.00      1.00      1.00     56962
```

	Metrica	Valore
0	Precision	0.946667
1	Recall	0.724490
2	F1 Score	0.820809
3	Accuratezza Test	0.999456
4	Accuratezza Train	0.999478

- **GradientBoostingClassifier:** Infine, abbiamo sperimentato il GBC che costruisce gli alberi in modo sequenziale, ognuno correggendo gli errori dei precedenti.

  Gli iperparametri dell'algoritmo utilizzati sono: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200} e ne risultano punteggi non assumibili come soddisfacenti in quanto tutti minori di 0.75 per le transazioni fraudolente.

```
GradientBoostingClassifier Classification_report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.74      0.60      0.66        98

    accuracy                           1.00     56962
   macro avg       0.87      0.80      0.83     56962
weighted avg       1.00      1.00      1.00     56962
```

	Metrica	Valore
0	Precision	0.737500
1	Recall	0.602041
2	F1 Score	0.662921
3	Accuratezza Test	0.998947
4	Accuratezza Train	0.999184

• **Regressione Logistica**

La regressione logistica è computazionalmente efficiente, il che significa che può essere addestrata su grandi dataset in tempi relativamente brevi. Questo è vantaggioso nel nostro caso, ma la metrica di recall calcolata è relativamente bassa e non soddisfacente.

Il parametro 'C' utilizzato rappresenta il parametro di regolarizzazione, noto come parametro di penalizzazione (Valori più alti di 'C' indicano una regolarizzazione più debole). 'penalty': specifica il tipo di regolarizzazione da applicare in questo caso L2, che aggiunge il quadrato dei pesi al termine di perdita. 'solver' specifica l'algoritmo utilizzato per ottimizzare i pesi del modello, l''lbfgs' nel nostro caso.

```
LogisticRegression Classification_report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.86      0.58      0.70        98

    accuracy                           1.00     56962
   macro avg       0.93      0.79      0.85     56962
weighted avg       1.00      1.00      1.00     56962
```

	Metrica	Valore
0	Precision	0.863636
1	Recall	0.581633
2	F1 Score	0.695122
3	Accuratezza Test	0.999122
4	Accuratezza Train	0.999232

• **Gaussian Naive Bayes**

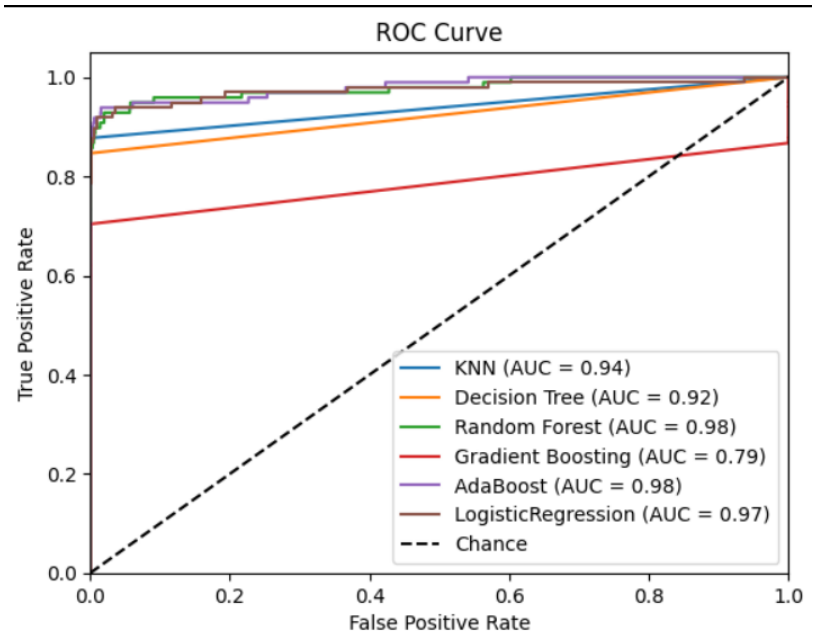
Il motivo principale dietro questa scelta è stata la necessità di gestire dati continui, come ad esempio l'importo delle transazioni e altre feature numeriche presenti nel nostro dataset.

```
GaussianNB Classification_report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56864
           1       0.06      0.82      0.11        98

    accuracy                           0.98     56962
   macro avg       0.53      0.90      0.55     56962
weighted avg       1.00      0.98      0.99     56962
```

	Metrica	Valore
0	Precision	0.060377
1	Recall	0.816327
2	F1 Score	0.112439
3	Accuratezza Test	0.977827
4	Accuratezza Train	0.978468

In questo caso notiamo che il valore di Precision è molto basso (circa 0.060), il che significa che il modello ha una tendenza a identificare come fraudolente molte transazioni che in realtà non lo sono. Potrebbe essere considerato un alto tasso di falsi positivi, anche se il valore di recall di circa 0.816 indica che il modello è in grado di individuare una buona parte delle transazioni effettivamente fraudolente nel dataset.

• **Conclusioni sull'addestramento dei modelli sul dataset completo**



Valutando invece le metriche AUC, che sarebbero le aree sottostanti alle curve ROC (Receiver Operating Characteristic) che misurano la capacità di un modello di classificazione di distinguere tra le due classi (positiva e negativa) in base alla sua capacità di regolare la soglia di classificazione, possiamo affermare che il RandomForest è il migliore modello in relazione anche alle metriche analizzate in precedenza. Per

quanto questo modello possa essere performante, è anche uno dei più complessi e non escluso dal ragionamento spiegato successivamente.

Nel corso dell'addestramento dei modelli di apprendimento, abbiamo notato una situazione di overfitting, indicata da valori costantemente pari a 1 per la maggior parte delle metriche nell'assegnazione della classe 0 alle transazioni legittime, a differenza dei valori molto più bassi nell'assegnazione della classe 1 alle transazioni fraudolente. Questo fenomeno ha evidenziato ciò che ci aspettavamo di trovare addestrando i modelli di apprendimento in un dataset così sbilanciato.

Successivamente dunque vedremo una delle due soluzioni progettate inizialmente per contrastare l'overfitting.

Addestramento modelli sul subsample del Dataset

• **Introduzione**

Qui di seguito andremo ad addestrare gli stessi modelli e ad analizzare gli stessi risultati delle metriche precedentemente descritte, ma questa volta sul subsample, in modo da avere un set di dati che consenta ai modelli di apprendimento di apprendere in modo più equo dalle due classi.

• **KNN**

Migliori iperparametri: {'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'uniform'}

```
KNeighborsClassifier Classification_report:
              precision    recall  f1-score   support

           0       0.89      0.96      0.93       110
           1       0.95      0.85      0.90        87

    accuracy                           0.91       197
   macro avg       0.92      0.91      0.91       197
weighted avg       0.92      0.91      0.91       197
```

	Metrica	Valore
0	Precision	0.948718
1	Recall	0.850575
2	F1 Score	0.896970
3	Accuratezza Test	0.913706
4	Accuratezza Train	0.964422

• **Alberi di decisione**

- **DecisionTreeClassifier**:

  Migliori iperparametri: {'max_depth': 3, 'min_samples_split': 2}

```
DecisionTreeClassifier Classification_report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94       110
           1       1.00      0.85      0.92        87

    accuracy                           0.93       197
   macro avg       0.95      0.93      0.93       197
weighted avg       0.94      0.93      0.93       197
```

	Metrica	Valore
0	Precision	1.000000
1	Recall	0.850575
2	F1 Score	0.919255
3	Accuratezza Test	0.934010
4	Accuratezza Train	0.944091

- **RandomForest**:

  Migliori iperparametri: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 300}

```
RandomForestClassifier Classification_report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94       110
           1       1.00      0.84      0.91        87

    accuracy                           0.93       197
   macro avg       0.94      0.92      0.93       197
weighted avg       0.94      0.93      0.93       197
```

	Metrica	Valore
0	Precision	1.000000
1	Recall	0.839080
2	F1 Score	0.912500
3	Accuratezza Test	0.928934
4	Accuratezza Train	0.989835

- **AdaBoost:**

  Migliori iperparametri: {'algorithm': 'SAMME.R', 'base_estimator': None, 'learning_rate': 0.5, 'n_estimators': 200}

```
AdaBoostClassifier Classification_report:
              precision    recall  f1-score   support

           0       0.91      0.97      0.94       110
           1       0.96      0.87      0.92        87

    accuracy                           0.93       197
   macro avg       0.93      0.92      0.93       197
weighted avg       0.93      0.93      0.93       197
```

	Metrica	Valore
0	Precision	0.962025
1	Recall	0.873563
2	F1 Score	0.915663
3	Accuratezza Test	0.928934
4	Accuratezza Train	1.000000

- **GradientBoostingClassifier**:

  Migliori iperparametri: {'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 100}

```
GradientBoostingClassifier Classification_report:
              precision    recall  f1-score   support

           0       0.90      0.98      0.94       110
           1       0.97      0.86      0.91        87

    accuracy                           0.93       197
   macro avg       0.94      0.92      0.93       197
weighted avg       0.93      0.93      0.93       197
```

	Metrica	Valore
0	Precision	0.974026
1	Recall	0.862069
2	F1 Score	0.914634
3	Accuratezza Test	0.928934
4	Accuratezza Train	1.000000

- **Regressione Logistica**

  Migliori iperparametri: {'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}

```
LogisticRegression Classification_report:
              precision    recall  f1-score   support

           0       0.90      0.98      0.94       110
           1       0.97      0.86      0.91        87

    accuracy                           0.93       197
   macro avg       0.94      0.92      0.93       197
weighted avg       0.93      0.93      0.93       197
```

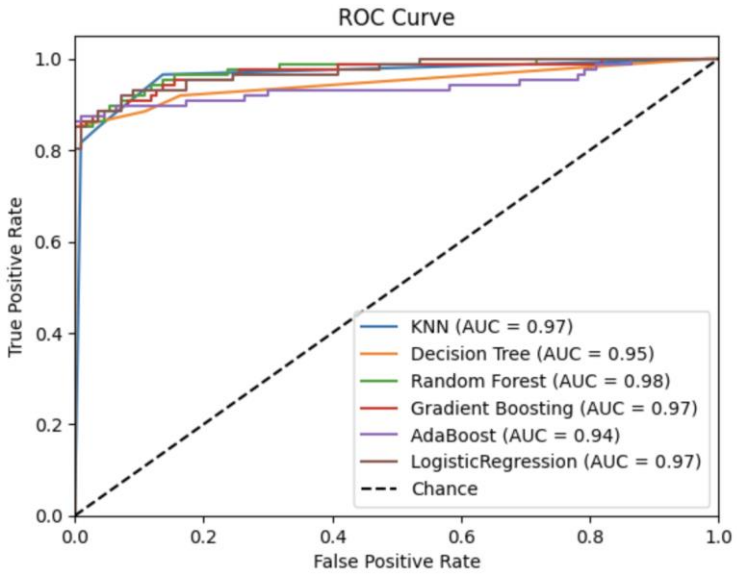
	Metrica	Valore
0	Precision	0.974026
1	Recall	0.862069
2	F1 Score	0.914634
3	Accuratezza Test	0.928934
4	Accuratezza Train	0.960610

- **Naive Bayes**

```
GaussianNB Classification_report:
              precision    recall  f1-score   support

           0       0.88      0.95      0.91       110
           1       0.94      0.83      0.88        87

    accuracy                           0.90       197
   macro avg       0.91      0.89      0.90       197
weighted avg       0.90      0.90      0.90       197
```

	Metrica	Valore
0	Precision	0.935065
1	Recall	0.827586
2	F1 Score	0.878049
3	Accuratezza Test	0.898477
4	Accuratezza Train	0.960610

• **Conclusioni sull'addestramento dei modelli sul Subsample:**



I risultati ottenuti dall'addestramento dei modelli sul subsample sono stati notevoli, come indicato dalle metriche di valutazione. In particolare, le metriche rivelano che i modelli hanno dimostrato una notevole abilità nel rilevare con successo le transazioni fraudolente senza compromettere la capacità di distinguere le transazioni legittime. In particolare, si sottolineano i risultati del modello DecisionTree e del modello RandomForest, con addirittura la precision per le transazioni fraudolente uguale a 1.00.

La precision, che misura la frazione di transazioni identificate come fraudolente che sono effettivamente fraudolente, è stata notevolmente alta, suggerendo una bassa incidenza di falsi positivi. Allo stesso tempo, il recall, che misura la frazione di transazioni fraudolente identificate correttamente, è stato altrettanto impressionante, indicando una capacità di rilevamento delle frodi molto efficace in quanto le transazioni legittime in tutti i modelli hanno Recall >0.91 (uguale a 1.00 nel DecisionTree e RandomForest), e tutte le transazioni fraudolente con Recall >0.83.

Addestramento modelli sui fold della Stratified Cross Validation

• **Modelli utilizzati:**

KNN - Cross-Validation AUC: 0.93

	Metrica	Valore
0	Precision	0.935206
1	Recall	0.774418
2	F1 Score	0.846857
3	Accuratezza	0.999515

Decision Tree - Cross-Validation AUC: 0.89

	Metrica	Valore
0	Precision	0.742686
1	Recall	0.778479
2	F1 Score	0.759562
3	Accuratezza	0.999147

Random Forest - Cross-Validation AUC: 0.95

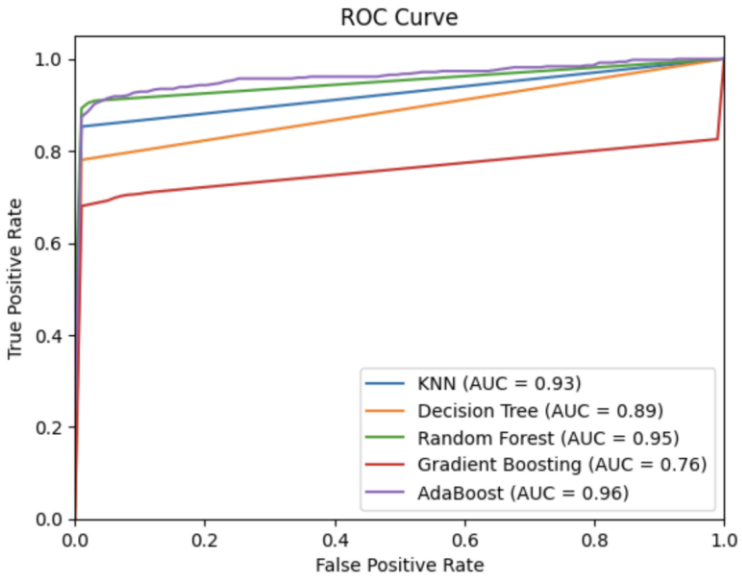
	Metrica	Valore
0	Precision	0.950816
1	Recall	0.780519
2	F1 Score	0.857196
3	Accuratezza	0.999551

Gradient Boosting - Cross-Validation AUC: 0.76

	Metrica	Valore
0	Precision	0.768840
1	Recall	0.603422
2	F1 Score	0.661460
3	Accuratezza	0.999024

AdaBoost - Cross-Validation AUC: 0.96

	Metrica	Valore
0	Precision	0.808880
1	Recall	0.687013
2	F1 Score	0.742101
3	Accuratezza	0.999175



Naive Bayes:

	Metrica	Valore
0	Precision	0.061812
1	Recall	0.829252
2	F1 Score	0.115038
3	Accuratezza	0.977918

• **Conclusioni sull'addestramento dei modelli con la stratified cross validation:**

In questo ultimo caso abbiamo ottenuto dei buoni ma non eccellenti risultati. I valori sono in generale abbastanza equilibrati tra loro. Alcuni modelli, come il KNN hanno un'accuratezza elevata ma un valore di Recall relativamente medio-basso. Ciò indica che potrebbero mancare nel rilevare tutte le transazioni fraudolente. In quasi tutti i modelli, inoltre, l'accuratezza del training è elevata, ma l'accuratezza del test è molto bassa rispetto ad essa, soprattutto nel Gradient Boosting. Questo rappresenta un chiaro caso di Overfitting.

Nel Naive Bayes invece notiamo risultati molto scarsi in quanto molti dei casi predetti come positivi sono falsi positivi. Allo stesso tempo, il modello ha un Recall abbastanza alto, il che indica che sta catturando una buona parte dei casi positivi nel dataset. Il risultato è un modello non affidabile e sbilanciato come indica la metrica F1.

In generale, i modelli che hanno mostrato risultati migliori sono il K-nn e il Random Forest.

7. Neural Network

Abbiamo scelto di utilizzare le reti neurali artificiali (NN) nel nostro caso di studio per diverse ragioni. Il nostro dataset contiene feature complesse e interdipendenti, come le caratteristiche delle transazioni finanziarie, e le NN sono in grado di apprendere rappresentazioni complesse da tali dati. Inoltre, queste

reti sono altamente scalabili e possono gestire grandi volumi di dati, il che è essenziale per analizzare transazioni. Le NN possono apprendere in modo profondo, identificando pattern non visibili con metodi di apprendimento superficiale. In sintesi, esse rappresentano una solida scelta per affrontare il complesso problema dell'individuazione di transazioni fraudolente nei dati finanziari.

Abbiamo scelto di implementare le NN con TensorFlow e Keras, per affrontare il nostro problema di rilevamento di transazioni fraudolente.

La nostra rete neurale è stata progettata nel seguente modo:

- **Strato di Input:** Questo strato ha un numero di unità uguale al numero di feature nel nostro dataset. Abbiamo scelto di utilizzare l'attivazione "relu" per questo strato.

- **Strati Nascosti:** Abbiamo scelto di utilizzare 2 layers da 10 e 8 unità. Rimandiamo la discussione di questa scelta alle prossime righe.

- **Strato di Output:** L'ultimo strato ha una sola unità, il che è adatto per un problema di classificazione binaria come il nostro. L'attivazione "sigmoid" è stata scelta in modo che l'output sia una probabilità tra 0 e 1, che può essere interpretata come la probabilità che una transazione sia fraudolenta.

Abbiamo utilizzato l'ottimizzatore "SGD"(stochastic gradient descending) e la funzione di perdita "binary_crossentropy". Queste scelte sono comuni per problemi di classificazione binaria.

La funzione di perdita "binary_crossentropy" è una misura comune della discrepanza tra le previsioni del modello e le etichette reali per problemi di classificazione binaria. Questa funzione di perdita è spesso utilizzata quando si ha una classificazione binaria, cioè quando l'obiettivo è prevedere una delle due classi (come nel nostro caso "fraudolento" o "legittimo").

SGD è un ottimizzatore di base ed è utilizzato in molte applicazioni di apprendimento automatico. Provando alcuni tassi di apprendimento, come 0.1 e 0.01, abbiamo notato che, il tasso di apprendimento ottimale per il dataset completo è 0.01 perché potrebbe divergere con tassi di apprendimento più alti. Al contrario, nel subsample il tasso di apprendimento migliore è 0.1. Ciò è confermato anche dalle metriche risultanti.

Rete Neurale sul Subsample (Stochastic Gradient Descent) con 3 hiddel layer e tasso di apprendimento: 0.01

```
9/9 [==============================] - 0s 3ms/step - loss: 0.1653
Epoch 95/100
9/9 [==============================] - 0s 2ms/step - loss: 0.1648
Epoch 96/100
9/9 [==============================] - 0s 3ms/step - loss: 0.1642
Epoch 97/100
9/9 [==============================] - 0s 3ms/step - loss: 0.1632
Epoch 98/100
9/9 [==============================] - 0s 3ms/step - loss: 0.1627
Epoch 99/100
9/9 [==============================] - 0s 3ms/step - loss: 0.1623
Epoch 100/100
9/9 [==============================] - 0s 3ms/step - loss: 0.1610
5/5 [==============================] - 0s 6ms/step
Report di classificazione per il validation set:
              precision    recall  f1-score   support

           0       0.85      0.97      0.90        62
           1       0.97      0.86      0.91        76

    accuracy                           0.91       138
   macro avg       0.91      0.91      0.91       138
weighted avg       0.91      0.91      0.91       138

10/10 [==============================] - 0s 3ms/step
Report di classificazione per il test set:
              precision    recall  f1-score   support

           0       0.90      1.00      0.95       148
           1       1.00      0.89      0.94       148

    accuracy                           0.94       296
   macro avg       0.95      0.94      0.94       296
weighted avg       0.95      0.94      0.94       296
```

Rete Neurale sul Subsample (Stochastic Gradient Descent) con 3 hidden Layer e tasso di apprendimento: 0.1

```
Epoch 95/100
9/9 [==============================] - 0s 2ms/step - loss: 0.1092
Epoch 96/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0995
Epoch 97/100
9/9 [==============================] - 0s 2ms/step - loss: 0.1086
Epoch 98/100
9/9 [==============================] - 0s 2ms/step - loss: 0.1085
Epoch 99/100
9/9 [==============================] - 0s 2ms/step - loss: 0.1025
Epoch 100/100
9/9 [==============================] - 0s 2ms/step - loss: 0.1023
5/5 [==============================] - 0s 2ms/step
Report di classificazione per il validation set:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95        71
           1       0.97      0.93      0.95        67

    accuracy                           0.95       138
   macro avg       0.95      0.95      0.95       138
weighted avg       0.95      0.95      0.95       138

10/10 [==============================] - 0s 1ms/step
Report di classificazione per il test set:
              precision    recall  f1-score   support

           0       0.89      0.99      0.94       152
           1       0.98      0.88      0.93       144

    accuracy                           0.93       296
   macro avg       0.94      0.93      0.93       296
weighted avg       0.94      0.93      0.93       296
```

Per questo motivo, ovvero il valore fisso di tasso di apprendimento, abbiamo utilizzato anche l'ottimizzatore "rmsprop" il quale è un miglioramento rispetto a SGD grazie all'adattamento dinamico del tasso di apprendimento, che lo rende più stabile e adatto per problemi di reti neurali.

L'ottimizzatore RMSprop è un algoritmo di ottimizzazione ampiamente utilizzato e utilizza una tecnica adattativa per regolare il tasso di apprendimento durante l'addestramento. Questo significa che il tasso di apprendimento viene adattato dinamicamente per ogni parametro del modello in base al gradiente storico delle iterazioni precedenti.

Rete Neurale su tutto il dataset (Root Mean Square Propagation):

```
Epoch 100/100
2493/2493 [==============================] - 3s 1ms/step - loss: 0.0052
1247/1247 [==============================] - 1s 1ms/step
Report di classificazione per il validation set:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     39800
           1       0.78      0.85      0.82        73

    accuracy                           1.00     39873
   macro avg       0.89      0.92      0.91     39873
weighted avg       1.00      1.00      1.00     39873

2671/2671 [==============================] - 3s 1ms/step
Report di classificazione per il test set:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85296
           1       0.77      0.77      0.77       147

    accuracy                           1.00     85443
   macro avg       0.89      0.88      0.89     85443
weighted avg       1.00      1.00      1.00     85443
```

L'addestramento è stato eseguito per 100 epoche con un batch size di 64.

Ritorniamo ora sulla scelta del numero di hidden layers. Abbiamo fatto questa scelta in quanto ci troviamo davanti ad un dataset complesso e con molte feature. Effettuando due prove con 3 e 2 layers rispettivamente da 10, 8 e 6 unità e da 10 e 8 unità, abbiamo notato che i risultati delle metriche sono abbastanza simili, tranne qualche valore che diminuisce/aumenta tra validation e test set. La differenza maggiore che abbiamo notato è quella nella funzione di perdita che aumenta con l'aumentare degli strati nascosti. Dunque, a parità di risultati delle metriche, scegliamo la NN con 2 hidden layers la quale ha una complessità minore.

In questo modo, abbiamo configurato e addestrato con successo una rete neurale che può apprendere dalle nostre feature di input e fare previsioni sulla probabilità di una transazione essere fraudolenta o legittima. Le scelte fatte durante la progettazione della rete sono il risultato di un processo di ottimizzazione mirato a massimizzare le prestazioni del modello sui nostri dati.

Rete Neurale su tutto il dataset (Stochastic Gradient Descent) con 2 Hidden Layer e tasso d'apprendimento: 0,01:

```
Epoch 99/100
2493/2493 [==============================] - 5s 2ms/step - loss: 0.0023
Epoch 100/100
2493/2493 [==============================] - 5s 2ms/step - loss: 0.0023
1247/1247 [==============================] - 2s 2ms/step
Report di classificazione per il validation set:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     39796
           1       0.94      0.79      0.86        77

    accuracy                           1.00     39873
   macro avg       0.97      0.90      0.93     39873
weighted avg       1.00      1.00      1.00     39873

2671/2671 [==============================] - 4s 2ms/step
Report di classificazione per il test set:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85284
           1       0.87      0.77      0.82       159

    accuracy                           1.00     85443
   macro avg       0.93      0.89      0.91     85443
weighted avg       1.00      1.00      1.00     85443
```

Tenendo in considerazione le molteplici prove effettuate secondo le diverse caratteristiche precedentemente descritte, le migliori metriche sono date da:

- Rete Neurale sul Subsample (Stochastic Gradient Descent) con 2 Hidden Layer e tasso di apprendimento: 0.1
- Rete Neurale su tutto il dataset (Stochastic Gradient Descent) con 2 Hidden Layer e tasso d'apprendimento: 0,01

Avendo risultati molto più omogenei fra tutte le prove, rispetto a tutti i modelli probabilistici precedentemente addestrati.

8. Conclusioni

Nel corso del nostro studio, abbiamo esplorato varie metodologie di apprendimento supervisionato, tra cui modelli di alberi decisionali, Random Forest, AdaBoost, Gradient Boosting, K-nn e Naive Bayes, al fine di affrontare il problema dell'individuazione di transazioni fraudolente. Tuttavia, alla luce dei risultati ottenuti, è emerso che la rete neurale addestrata con i dati appena citati ha dimostrato prestazioni superiori rispetto agli altri modelli valutati. Sicuramente, questo stesso lavoro si potrebbe effettuare su un Dataset con informazioni in chiaro così da evidenziare ipotetiche relazioni tra le features, andando a variarne gli sviluppi.