Progetto Ingegneria della conoscenza – Forecasting the Success of Bank Marketing Calls: A Predictive Analysis

Repo: https://github.com/FValerio96/bank-marketing.git

Flavio Valerio, matricola 708928,

f.valerio6@studenti.uniba.it

Anno accademico: 2023-2024

Presentazione caso di studio:

Introduzione

Il progetto seguente è volto a mettere in pratica ciò che si è appreso durante il corso di ingegneria della conoscenza, in modo compatibile alle tempistiche e alle risorse a disposizione, cercando di effettuare supposizioni su quanto osservato dal comportamento dei modelli usati e osservazioni sui risultati ottenuti.

Il caso di studio vede, dopo una prima fase di pulizia del dataset, l'applicazione dei modelli *k-NN*, *Random Forest*, *adaBoost*, *XGBoost* seguendo il seguente approccio:

1. Utilizzo dei modelli citati sul dataset originale per ottenere dei risultati "base"
2. Applicazione del undersampling sul dataset, in quanto l'outcome risulta essere sbilanciato.
    a. Oversampling scartato per via della differenza elevata tra classe minoritaria e maggioritaria, che potrebbe portare ad allontanarsi dalla realtà osservata riducendo la credibilità dei risultati ottenuti.

Si è successivamente eseguito un'applicazione dell'apprendimento probabilistico tramite classificatore *naive bayes* gaussiana con analisi dei risultati.

Dataset

Il dataset possiede informazioni prese dal UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/bank+marketing. Le informazioni sono raccolte durante una campagna marketing telefonica, con anche più di un contatto con lo stesso cliente. Il dataset utilizzato nel caso di studio contiene un buon numero di features di input (20) ed è relativamente grande per le risorse a disposizione (40.000 esempi).

Il dataset è composto da molteplici features categorical, le quale attraverso un dizionario sono state mappate ad un corrispondente numero per permettere il lavoro dei modelli citati in precedenza.

Il dataset presenta una variabile outcome binaria, presenta dunque due classi, che indicano se il telefonato ha accettato o meno di aprire il deposito bancario. Tale variabile presenta uno squilibrio importate, con 9/10 degli esempi su una sola delle due classi.

In seguito, cercherò di sottolineare come mi abbiamo influenzato queste informazioni nella scelta di progetto.

Argomenti d'esame trattati nel progetto

- Apprendimento supervisionato
- Ragionamento probabilistico

i modelli usati sono quelli già citati.

Creazione del Dataframe:

considerazioni su feature e valori unkown nelle stesse

Il dataframe selezionato presentata una feature ‘duration’ (durata della call) che influenza fortemente il target di output (ad esempio, se duration=0 allora y='no'). Tuttavia, la durata non è nota prima che venga eseguita una chiamata. Inoltre, dopo la fine della chiamata, y è ovviamente noto. Pertanto, questo input dovrebbe essere escluso se l'intenzione è quella di avere un modello predittivo, come nel nostro caso.

Data la presenza di valori unkown all'interno del dataset, si valuta la possibilità di rimuovere le righe che li contengono data la percentuale tendenzialmente esigua, tranne che per default dove si arriva ad un 20.87%. Riflettendo sul ruolo di questi valori unkown nel dataset, ci si rende conto che unkown è un’informazione utile, in quanto l'utente che riceve la telefonata marketing sceglie se dare o meno le informazioni che lo riguardano, che poi ritroviamo nel dataset. Unkown viene ritenuta un’informazione circa l'engagement del potenziale cliente, quindi utile nel contesto di predizione di conversione delle telefonate nell'apertura del deposito.

features del dataset

Il dataset presenta le seguenti variabili in input divisibili in categorie:

Dati del cliente della banca

- age: (numerica)
- job: tipo di lavoro (categorico)
- marital: stato civile (categorico)
- education (categorico)
- default: ha un credito insoluto? (categorico)
- housing: ha un prestito per l'abitazione? (categorico)
- loan: ha un prestito personale? (categorico)

Dati Relativi all'ultimo contatto della campagna in corso

- contatto: tipo di comunicazione del contatto (categorico)
- mese: mese dell'anno dell'ultimo contatto (categorico)
- giorno_della_settimana: giorno della settimana dell'ultimo contatto (categorico)
- duration: durata dell'ultimo contatto, in secondi (numerico)

Altri attributi

- campagna: numero di contatti effettuati durante questa campagna e per questo cliente (nume rico, include l'ultimo contatto)
- pdays: numero di giorni trascorsi dopo che il cliente è stato contattato per l'ultima volta da un a campagna precedente (numerico; 999 significa che il cliente non è stato contattato in prece denza)
- precedente: numero di contatti effettuati prima di questa campagna e per questo cliente (num erico)
- risultato: risultato della campagna di marketing precedente (categorico).

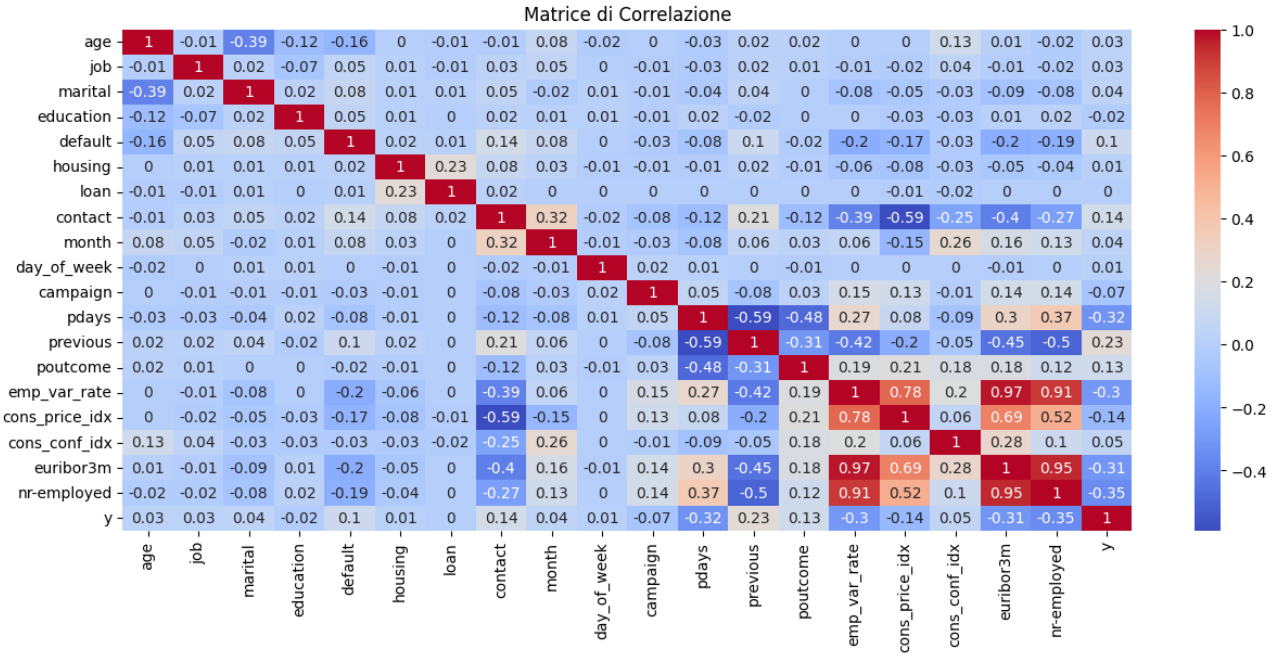
Attributi del contesto sociale ed economico

- emp.var.rate: tasso di variazione dell'occupazione - indicatore trimestrale (numerico)
- cons.price.idx: indice dei prezzi al consumo - indicatore mensile (numerico)
- cons.conf.idx: indice di fiducia dei consumatori - indicatore mensile (numerico)
- euribor3m: tasso euribor a 3 mesi - indicatore giornaliero (numerico)
- nr.occupati: numero di dipendenti - indicatore trimestrale (numerico)

matrice di correlazione

Una **matrice di correlazione** è una tabella che mostra i coefficienti di correlazione tra molte variabili. Ogni cella nella tabella mostra la correlazione tra due variabili. Un coefficiente di correlazione varia tra -1 e 1, avvicinarsi ad uno dei due estremi significa avere una correlazione molto alta, la differenza è che:

- 1 correlazione positiva
- -1 correlazione negativa (proporzionali in modo inverso)



Possiamo notare delle correlazioni fortemente positive (>0.8)

- emp_var_rate e euribor3m (0,97)
- nr_employed e euribor3m (0,95)
- nr_employed e emp_var_rate (0,91)

In sintesi, queste correlazioni suggeriscono che gli attributi del contesto sociale ed economico sono strettamente collegati e possono variare insieme in risposta a fattori economici e di mercato. Ad esempio, cambiamenti nel mercato del lavoro potrebbero influenzare i tassi di interesse o viceversa.

Buone Correlazioni Negative (< -0,5):

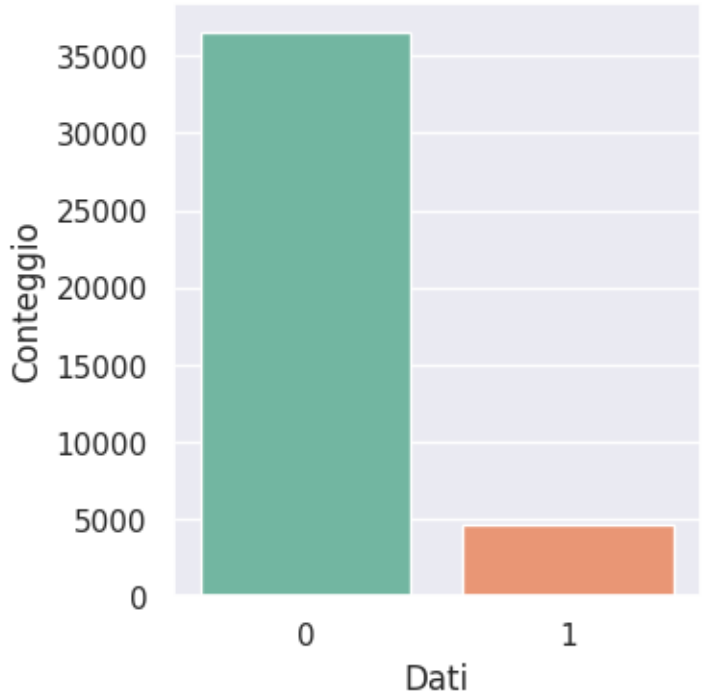
- previous e pdays (-0,59)

- cons_price_idx e contact (-0,59)
- nr_employed e previous (-0,5)

la correlazione negativa tra previous e pdays suggerisce che un aumento del numero di contatti effettuati prima di questa campagna è associato a una diminuzione del numero di giorni trascorsi dall'ultimo contatto precedente, Questo potrebbe indicare che i clienti che sono stati contattati più volte in precedenza tendono ad avere un periodo più breve tra i contatti, il che potrebbe indicare una maggiore attività di marketing da parte della banca o un interesse maggiore da parte del cliente.

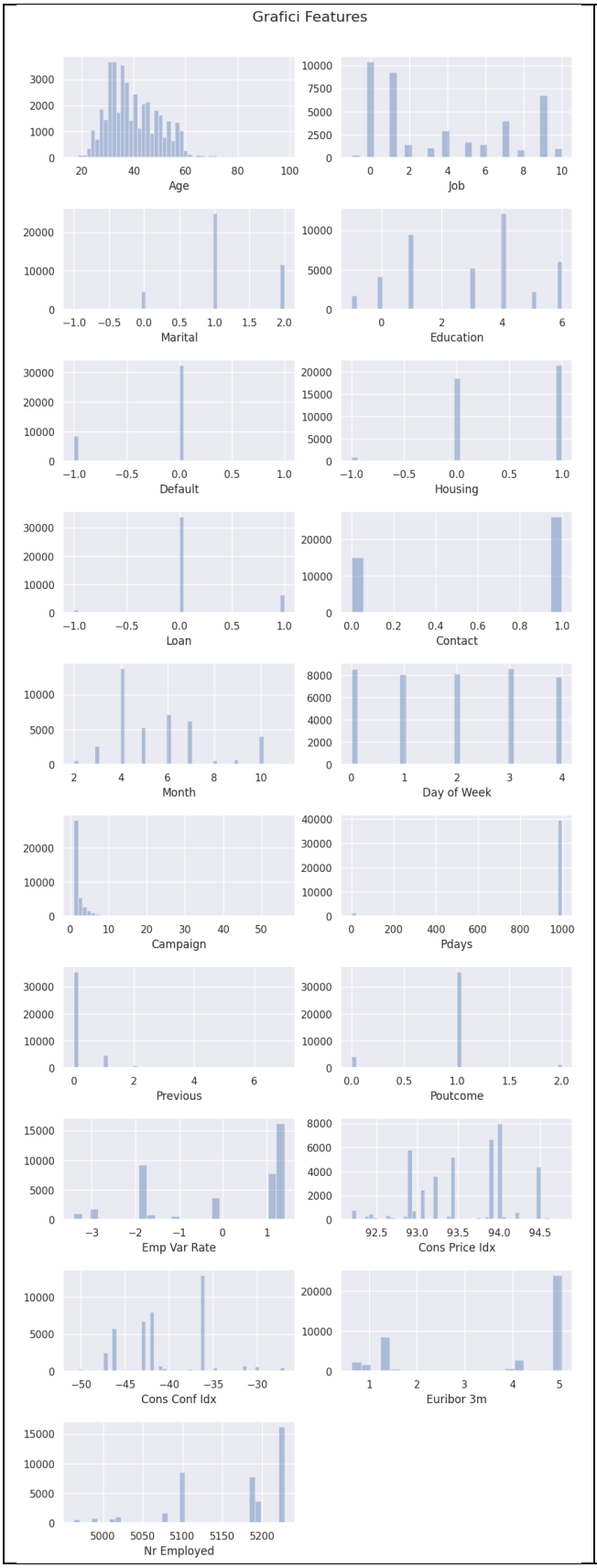
Grafici sul dataset:

outcome



Il grafico mostra i dati sulla features target y, che rappresenta la scelta del telefonato dove 0 indica il rifiuto e 1 l'apertura del deposito. È immediato notare, che stiamo lavorando con un dataset che si presenta sbilanciato. Riflettendo sul contesto del dataset, ciò era auspicabile, è improbabile che la metà dei telefonati scelga di favorire dell'offerta. Visionato questo grafico, si è compreso di aver la possibilità di sfruttare undersampling per creare un bilanciamento e comprendere se abbiamo un miglioramento delle prestazioni.
Sottolineo come lo sbilanciamento dell'outcome sia centrale nel caso di studio e approccio al progetto.

Features di input



Alcune osservazioni sui grafici a sinistra:

- Età: Il dataset mostra una distribuzione dell'età con un picco evidente tra i 30 e i 40 anni.
- Stato Civile: L'analisi dello stato civile rivela un predominio di individui sposati. Ciò probabilmente deriva dal fatto che la banca ha ritenuto più idonei come potenziali clienti persone sposate.
- Mese: Un'altra osservazione interessante riguarda il mese di aprile, che mostra un picco notevole
- Pdays: Infine, la variabile 'pdays', che rappresenta il numero di giorni trascorsi dall'ultimo contatto, mostra un'elevata frequenza del valore "999". Questo valore è comunemente usato per indicare che non c'è stato alcun contatto precedente. Quindi, la maggior parte delle telefonate nel dataset rappresenta il primo contatto con l'individuo e potrebbe sottolineare la difficoltà nell'avere una seconda telefonata col potenziale cliente

Splitting stratificato

L'applicazione dello splitting stratificato permette di mantenere la stessa proporzione delle classi presenti nel dataset, quando andiamo ed effettuare la divisione tra training e dataset. Come detto, questo progetto vede il confronto tra le performance con dataset originale e la versione undersampled per stabilire un equilibrio. Al bivio, sull'effettuare prima splitting o undersampling, si è optato per il primo, questo perché:

- applicare tecniche di under/over-sampling prima dello splitting dei dati potrebbe consentire al modello di memorizzare esempi, causando overfitting sui dati di training e compromettendo la capacità di generalizzazione sui dati di test.
- l'uso di splitting stratificato prima di applicare tecniche di under/over-sampling può portare a modelli più generalizzabili, in grado di funzionare meglio su dati non visti, poiché sono stati addestrati su un set di training rappresentativo delle proporzioni reali delle classi nel dataset originale.

Apprendimento supervisionato, confronto tra modelli su dataset originale e undersampled

K-**Fold cross validation stratificata**

Partiamo dal presupposto, che il miglior modello non sia colui il quale ci offre le migliori performance nel training set, bensì nel test set ovvero, sui dati "nuovi" riuscendo inoltre ad evitare l'overfitting e/o underfitting. Si è mantenuta la distribuzione reale tra le classi d'interesse per evitare di generare folds d'esempi che portino a situazioni ottimali e ad un finto miglioramento delle prestazioni. Ad esempio, il dataset offre per *y* una classe maggioritaria ed una minoritaria. Costruendo un fold nel quale il ruolo di minoritaria e maggioritaria si invertono, avremo migliori prestazioni per la classe originariamente minoritaria, un risultato falso è lontano da ciò che si può aspettare.

Ottimizzazione: accuracy

Si è riflettuto, su quale performance ottimizzare. Come ben sappiamo gli errori non sempre hanno lo stesso impatto, nello studio di malattie un falso negativo e ben più grave di un falso positivo. Quindi, si è riflettuto sul caso in questione e si è ritenuto, però, che al contrario dell'esempio precedente sulla predizione di una vendita, falso positivo e falso negativo non hanno maggior priorità l'uno sull'altra. Dunque, ho deciso di ottimizzare l'accuracy, ovvero, la percentuale di predizioni corrette fatte dal modello rispetto le predizioni totali effettuate.

Grid search

La Grid Search è una strategia fondamentale per migliorare le prestazioni dei modelli di machine learning. Funziona esplorando un insieme di possibili combinazioni di parametri predefiniti e valutando quale combinazione produce le migliori performance del modello.

```
grid_search = GridSearchCV(knn_classifier, param_grid, scoring='accuracy', refit=True, cv=stratified_kfold)
```

L'esempio in figura riguarda il knn, ma ciò è valido per ogni modello usato.

Come anticipato, si è scelto di porre l'accento sull'accuracy e si è data questa informazione al grid search. Non ci si aspetta di ignorare le altre performance in virtù dell'accuracy, ma di porvi l'accento.

Per quanto riguarda i parametri, questi variano da modello in modello. Tuttavia, ci sono tre diversi modelli basati su alberi: random forest, xgBoost, adaBoost. Sui modelli citati vi sono parametri in comune, sui quali si è deciso di mantenere gli stessi valori al fine di poter effettuare confronti equi successivamente.

K-NN

K-Nearest Neighbors è un algoritmo di classificazione e regressione supervisionata. In fase di classificazione, per un dato punto di test, K-NN trova i K punti più vicini nel dataset di addestramento utilizzando una metrica di distanza (ad es. distanza euclidea) e assegna al punto la classe più frequente tra questi vicini. In fase di regressione, restituisce la media o la mediana dei valori target dei K punti più vicini come previsione per il punto di test.

I **parametri** usati per la grid search sul K-NN sono stati i seguenti:

```python
n_neighbors = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 50, 65, 70, 90, 110, 200]
weights = ['uniform']

# Parameters to test with GridSearchCV
param_grid = {
    'n_neighbors': n_neighbors,
    'weights': weights
}
```

Inizialmente il numero di n_neighbors era inferiore e con valore più basso. Tuttavia, agevolato dalla rapidità del modello e dopo aver visto il grafico (mostrato in seguito), che mostrava un aumento dell'accuracy all'aumentare dei vicini ho scelto di aumentare il numero di vicini

K-NN sul dataset originale

Come anticipato, il primo passo è stato l'applicazione dei modelli sul dataset originale. Per tanto, la grid search tra i parametri messi a disposizione, ha scelto i *best params* con i quali abbiamo addestrato il k-nn, con i seguenti risultati:

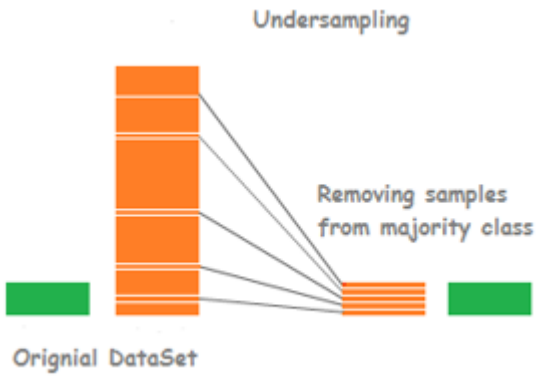
```
% ---------------- KNN su dati di test ---------------- %

Parametri ottimali: {'n_neighbors': 200, 'weights': 'uniform'}
              precision    recall  f1-score   support

           0       0.90      0.99      0.94      7303
           1       0.68      0.18      0.29       935

    accuracy                           0.90      8238
   macro avg       0.79      0.59      0.62      8238
weighted avg       0.88      0.90      0.87      8238


% ---------------- KNN su dati di training ---------------- %

Parametri ottimali: {'n_neighbors': 200, 'weights': 'uniform'}
              precision    recall  f1-score   support

           0       0.91      0.99      0.95     29245
           1       0.70      0.18      0.29      3705

    accuracy                           0.90     32950
   macro avg       0.81      0.59      0.62     32950
weighted avg       0.88      0.90      0.87     32950
```

Considerazioni sui dati:

- **Performance del modello**: Le prestazioni del modello sono influenzate dallo sbilanciamento delle classi nell' outcome. Ciò comporta che per sé per una classe riusciamo ad ottenere buoni risultati, per l'altra subiamo lo sbilanciamento. Approfondiamo con altre considerazioni.
- **Overfitting**: i risultati ottenuti in fase di testing e training sono estremamente simili, ciò non lascia presagire presenza di overfitting.
- **Recall** e **Precisione**: vediamo risultati diversi tra le due metriche
    - Classe 0: Per la classe 0, il recall è più alto rispetto alla precisione. Questo significa che il modello è più incline a classificare correttamente tutti i positivi reali (classe 0), ma potrebbe classificare erroneamente alcuni negativi (classe 1) come positivi (classe 0). In altre parole, il modello è efficace nel rilevare la classe 0, ma a volte può sovrastimare la sua presenza.
    - Classe 1: Per la classe 1, la precisione è più alta rispetto al recall. Questo indica che il modello è più cauto nel predire la classe 1: tende a classificare correttamente i negativi, ma potrebbe non rilevare tutti i positivi reali. In pratica, quando il modello predice la classe 1, è solitamente corretto.
- **numero** di **vicini**: il modello sembra performare al meglio con 200 vicini, (come possibile notare nel grafico posto dopo il k-nn undersampled, intitolato “accuracy vs n_ neighbors”), ciò rappresenta il picco dell’accuratezza, il che da un lato è un punto a favore. Tuttavia, ciò potrebbe anche significare che il modello è troppo semplificato, che non cattura la complessità nei dati e quindi **underfitting**.
- **k fold cross validation**: il valore iniziale di k era stato impostato a 10. Tuttavia, data la grandezza del dataset si è pensato di testare ulteriori folds. con un k = 20, seppure nell'ordine dei millesimi si è ottenuto un miglioramento del test, perdendo sempre nell'ordine dei millesimi in training. Per tanto si è ritenuto equivalente un k = 10 ad un k = 20.

K-NN con undersampling

Al fine di studiare le implicazioni nelle performance, dello sbilanciamento nell’outcome si è applicato l’undersampling per bilanciare, come mostrato in figura.



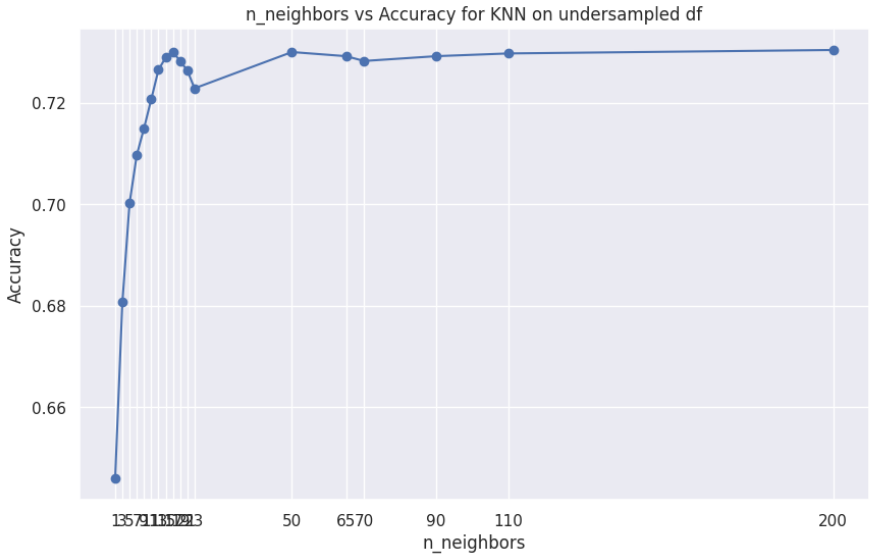
I risultati ottenuti sono i seguenti:

```
% ---------------- KNN on test data ---------------- %

Best parameters: {'n_neighbors': 200, 'weights': 'uniform'}
              precision    recall  f1-score   support

           0       0.94      0.84      0.89      7303
           1       0.32      0.61      0.42       935

    accuracy                           0.81      8238
   macro avg       0.63      0.72      0.65      8238
weighted avg       0.87      0.81      0.83      8238


% ---------------- KNN on training data with cross-validation ---------------- %

Best parameters: {'n_neighbors': 200, 'weights': 'uniform'}
              precision    recall  f1-score   support

           0       0.69      0.83      0.76      3705
           1       0.79      0.63      0.70      3705

    accuracy                           0.73      7410
   macro avg       0.74      0.73      0.73      7410
weighted avg       0.74      0.73      0.73      7410
```

È evidente fin da subito che l'uso dell'undersampling ha portato a un miglior bilanciamento tra le due classi, il che si riflette chiaramente nelle prestazioni del modello. Le differenze tra le performance delle due classi sono ora molto meno marcate rispetto al dataset originale. È importante notare che, sebbene la classe 0 abbia prestazioni inferiori rispetto al dataset originale, questa discrepanza potrebbe essere attribuita alla precedente sovrarappresentazione dei dati, che potrebbe aver influito su risultati ottimistici. D'altra parte, nell'approccio con undersampling, le prestazioni sono più affidabili e realistiche.

Il K- NN undersampled presenta un miglior f1-score nel test che nel training set. Ciò deriva dal fatto che il training set è undersampled, il test set, al contrario, no. La distribuzione del training set è test set sono diverse e per tanto abbiamo prestazioni diverse.

Si nutrono dei sospetti, inoltre per la presenza di sovradattamento del modello, probabilmente figlio della selezione di iperparametri ad opera della grid search. I sospetti nascono dal mantenimento di 200 come numero di vicini, nonostante il netto rimpicciolimento del dataset.



Il grafico su riportato mostra il rapporto tra numero di vicini scelto e accuracy ottenuta. Notiamo come sarebbe stato possibile scegliere un numero di vicini di 50, mantenendo un accuracy che dal grafico sembra essere approssimabile, se vediamo come errore accettabile un errore nell'ordine dei centesimi. In più, come detto, anche nel K-NN su dataset originale, Scegliendo 200 vicini, è probabile il modello non stia cogliendo la complessità dei dati, soprattutto ora che il dataset conta circa 1/4 degli esempi rispetto il dataset originale.

AdaBoost

AdaBoost utilizza il boosting, un approccio sequenziale, per migliorare la performance del modello. Durante il processo di addestramento, AdaBoost si concentra sugli esempi classificati erroneamente dai modelli precedenti, aumentando progressivamente la loro importanza. Costruisce una sequenza di classificatori deboli, ognuno dei quali viene addestrato per concentrarsi sugli errori dei modelli precedenti. Alla fine, combina i classificatori deboli in un unico classificatore forte tramite votazione ponderata.

AdaBoost, così come xgBoost e random forest appartengono alla famiglia dei modelli ensamble, in cui si prende un numero di modelli e combina le loro previsioni per fare una previsione per d'insieme.

Gli iperparametri messi a disposizione della grid search sono stati:

```
n_estimators = [10, 50, 100, 200, 250, 300]
# Parametri da testare con GridSearchCV per AdaBoost
param_grid = {
    'n_estimators': n_estimators,
    'learning_rate': [0.01, 0.1, 1.0],
}
```

Spiegazione degli iperparametri:

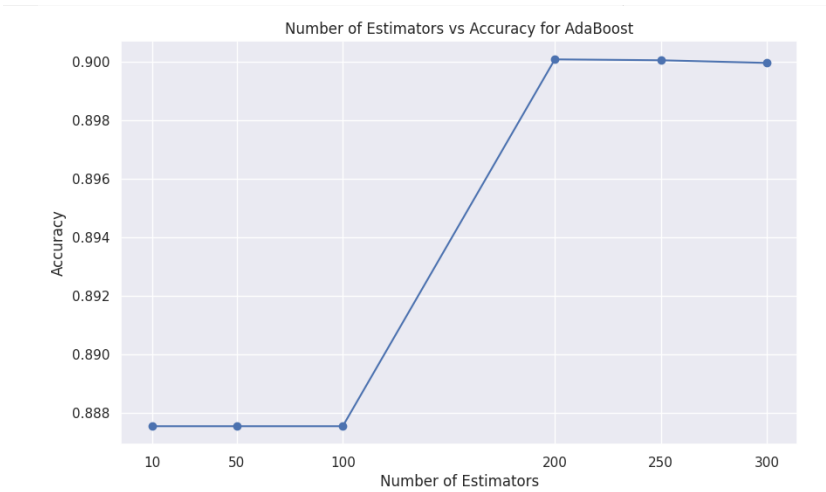
1. **n_estimators**: Questo parametro indica il numero di classificatori deboli (solitamente alberi decisionali poco profondi) che verranno utilizzati per comporre il classificatore forte. In altre parole, rappresenta il numero di iterazioni del processo di boosting. Aumentare il numero di stimatori può portare a un miglioramento delle prestazioni del modello, fino a un certo punto. Tuttavia, un numero troppo elevato di stimatori può portare all'overfitting.
2. **learning_rate**: Questo parametro controlla il contributo di ciascun classificatore debole nell'aggiornamento del modello complessivo. Più piccolo è il valore del learning rate, più lento è il processo di addestramento e più conservativo è l'aggiornamento dei pesi degli esempi classificati erroneamente. Un learning rate più piccolo può contribuire a ridurre il rischio di overfitting, ma potrebbe richiedere un numero maggiore di stimatori per ottenere prestazioni comparabili. Al contrario, un learning rate più grande accelera il processo di addestramento, ma potrebbe rendere il modello più sensibile al rumore nei dati di addestramento e all'overfitting.

AdaBoost sul dataset originale

Osserviamo i risultati ottenuti:

```
% ---------------- AdaBoost su dati di test ---------------- %

Parametri ottimali: {'learning_rate': 1.0, 'n_estimators': 300}
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      7303
           1       0.64      0.23      0.33       935

    accuracy                           0.90      8238
   macro avg       0.77      0.60      0.64      8238
weighted avg       0.88      0.90      0.88      8238


% ---------------- AdaBoost su dati di training ---------------- %

Parametri ottimali: {'learning_rate': 1.0, 'n_estimators': 300}
              precision    recall  f1-score   support

           0       0.91      0.99      0.95     29245
           1       0.67      0.24      0.35      3705

    accuracy                           0.90     32950
   macro avg       0.79      0.61      0.65     32950
weighted avg       0.88      0.90      0.88     32950
```

Come intuibile, anche qui l'assenza di bilanciamento tra le classi ha comportato risultati nettamente diversi tra classe 0 e 1. Tuttavia, bisogna dare atto all'adaBoost di aver ottenuto una buona precisione (64% in testing) nella classe minoritaria. L'assenza di un trade-off concreto tra training e testing non induce a pensare ci sia overfitting in questo caso. Nonostante ciò, le prestazioni sulla classe minoritaria non sono accettabili e ci si aspettano miglioramenti dopo l'undersampling.



Osservando il grafico è possibile notare come la grid search, abbia optato nuovamente per il picco d'accuracy, ovvero, 300 n_estimators. Tuttavia, non porta ad un concreto aumento dell'accuracy, come una lettura rapida del grafico potrebbe far credere. Il miglioramento ottenuto e di 0.012, al costo di un overfitting notevole. Ancor più, il grafico ci dice che il valore d'accuracy tra 10 e 100 n_estimators e pressoché lo stesso, portandomi a credere che potremmo combattere l'overfitting modificando nei parametri della grid search n_estimators portandolo a valori nell'intervallo [2,10].

AdaBoost con undersampling

```
% ---------------- AdaBoost on test data ---------------- %

Best parameters: {'learning_rate': 1.0, 'n_estimators': 250}
              precision    recall  f1-score   support

           0       0.95      0.84      0.89      7303
           1       0.34      0.64      0.45       935

    accuracy                           0.82      8238
   macro avg       0.65      0.74      0.67      8238
weighted avg       0.88      0.82      0.84      8238


% ---------------- AdaBoost on training data with cross-validation ---------------- %

Best parameters: {'learning_rate': 1.0, 'n_estimators': 250}
              precision    recall  f1-score   support

           0       0.71      0.84      0.77      3705
           1       0.80      0.66      0.72      3705

    accuracy                           0.75      7410
   macro avg       0.76      0.75      0.75      7410
weighted avg       0.76      0.75      0.75      7410
```

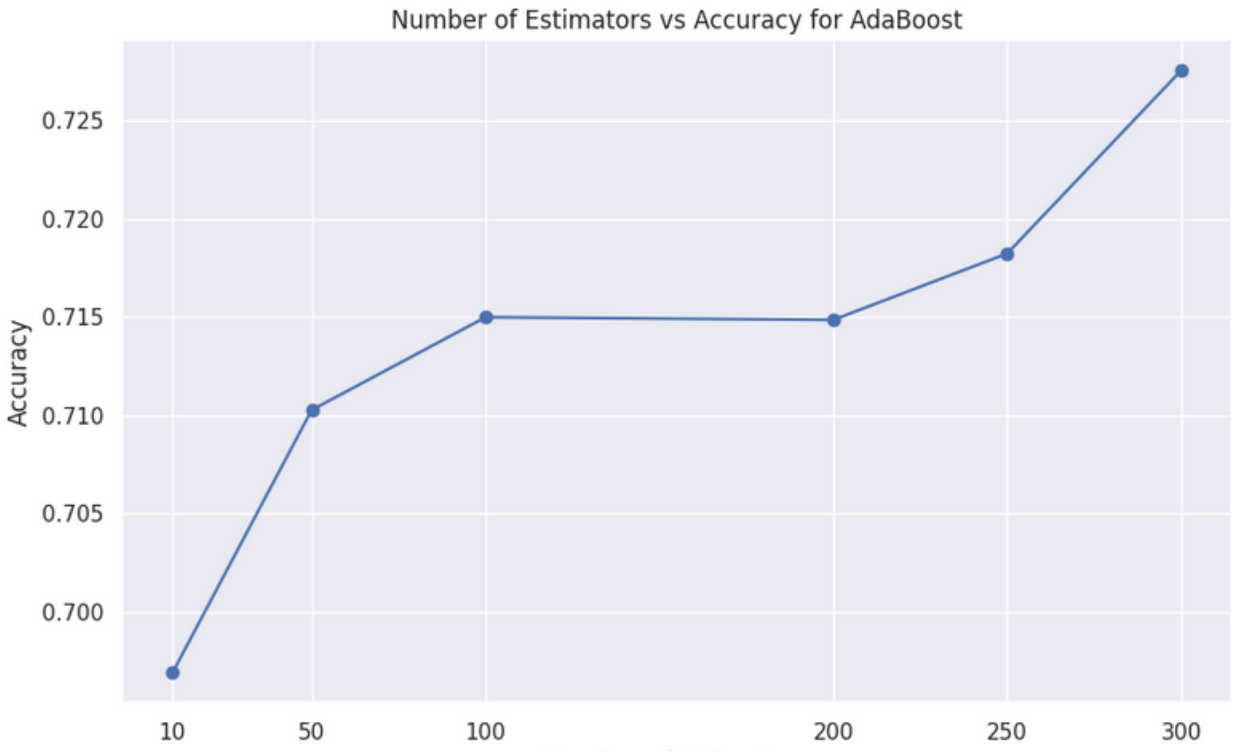
Impossibile non notare la presenza di overfitting.

il modello ha un'accuratezza del 82% sui dati di test e del 75% sui dati di addestramento con validazione incrociata. Questa differenza di performance indica che il modello potrebbe essere troppo complesso e quindi sovradattato ai dati di addestramento.

Inoltre, la precisione e il recall per la classe 1 sono significativamente più bassi sui dati di test rispetto ai dati di addestramento, il che è un altro indicatore di overfitting.

I dati ottenuti nel training sono nettamente migliori, questo ci dice che il modello su "dati nuovi" non riesce a performare.

Per approfondire il discorso, ho stampato il plot accuracy vs n_estimators:



il grafico ci mostra come in questo caso, la grid search non abbia scelto il picco di accuracy nonostante nel campo scoring avessimo selezionato accuracy. Probabilmente seppur prioritaria, la grid search ha cercato di equilibrare anche le altre performance.

Tra 10 estimators e 300 la differenza di accuracy è di 0.025, anche in questo caso risulta che diminuire la scelta di n_estimators per la grid search all'intervallo [2,10] potrebbe essere la scelta migliore, o quantomeno, un'ipotesi degna d'esser testata.

XgBoost

L'algoritmo di XGBoost si basa sul concetto di boosting, che combina un insieme di modelli deboli, tipicamente alberi decisionali, in modo sequenziale. Questi modelli vengono addestrati per correggere gli errori dei loro predecessori, creando un modello finale accurato.

Sono stati offerti, alla grid search, i seguenti parametri tra cui scegliere:

```
n_estimators = [10, 50, 100, 200, 250, 300]
learning_rates = [0.01, 0.1, 1.0]
max_depth = [6, 10, 20, 30, 40, 50, 75 , 100]

# Parametri da testare con GridSearchCV per XGBoost
param_grid = {
    'n_estimators': n_estimators,
    'learning_rate': learning_rates,
    'max_depth': max_depth
}
```

XgBoost su dataset originale

```
% ---------------- XGBoost su dati di test ---------------- %

Parametri ottimali: {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 50}
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      7303
           1       0.63      0.25      0.36       935

    accuracy                           0.90      8238
   macro avg       0.77      0.62      0.65      8238
weighted avg       0.88      0.90      0.88      8238


% ---------------- XGBoost su dati di training ---------------- %

Parametri ottimali: {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 50}
              precision    recall  f1-score   support

           0       0.91      0.98      0.95     29245
           1       0.67      0.27      0.38      3705

    accuracy                           0.90     32950
   macro avg       0.79      0.62      0.66     32950
weighted avg       0.89      0.90      0.88     32950
```

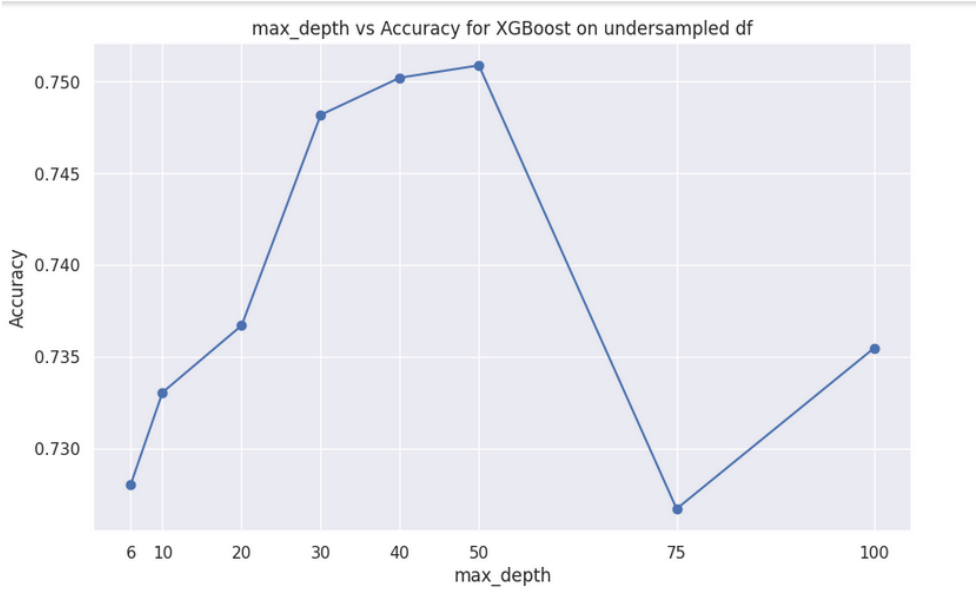
L’xgBoost sul dataset originale ha dato risultati molto simili all’adaBoost: ottime prestazioni nella classe maggioritaria, buona precisone per la minoritaria, che per il resto però, ha performance non accettabili. L’accuracy presenta ottimi valori.

XgBoost con undersampling

```
% ---------------- XGBoost su dati di addestramento ---------------- %

Parametri ottimali: {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 50}
              precision    recall  f1-score   support

           0       0.92      0.99      0.95     29245
           1       0.75      0.31      0.44      3705

    accuracy                           0.91     32950
   macro avg       0.84      0.65      0.69     32950
weighted avg       0.90      0.91      0.89     32950


% ---------------- XGBoost su dati di test ---------------- %

Parametri ottimali: {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 50}
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      7303
           1       0.63      0.25      0.36       935

    accuracy                           0.90      8238
   macro avg       0.77      0.62      0.65      8238
weighted avg       0.88      0.90      0.88      8238
```

Anche in questo caso come ipotizzabile, c'è stato un miglioramento delle prestazioni della classe minoritaria, ma non sensibili come nel caso del K-NN. L'ipotesi è che il dataset non risponda bene, all'aumento di complessità dei modelli. Questo potrebbe derivare da diversi fattori come la presenza di rumore nel dataset o un'eccessiva linearità di quest'ultimo.

sia AdaBoost che XGB overfittano gli esempi di training. In effetti, la teoria afferma che l'aumento della complessità di un modello aumenta la sua capacità di adattarsi ai dati legandosi agli esempi di training fino a diventare inefficace su esempi sconosciuti.



In questo caso la misura di complessità è data dalla profondità degli alberi che costituiscono l'XgBoost, lunghezza stabilita "ciecamente" dal GridSearch in quanto basatosi esclusivamente sullo score dell'accuracy. Tuttavia, se il GridSearch è configurato per massimizzare solo una metrica, come l'accuracy, senza considerare altre metriche o il rischio di overfitting, potrebbe selezionare una profondità dell'albero che porta al miglior risultato su quella metrica specifica sui dati di addestramento. Questo potrebbe comportare la selezione di modelli troppo complessi, che sono più inclini all'overfitting.

xgBoost con scale pos weight

per l'xgBoost si è provato anche a sfruttare un iperparametro di bilanciamento *scale_pos_weight* al quale è stato assegnato un valore pari al rapporto tra classe maggioritaria e minoritaria. Questo valore viene utilizzato per ridimensionare il peso delle istanze positive durante l'addestramento del modello.

L'idea era di testare questa possibilità in alternativa all'undersampling, mantenendo dunque invariato il numero di esempi, ma lavorando sui pesi.

```
% ---------------- XGBoost su dati di addestramento ---------------- %

Parametri ottimali: {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 50}
              precision    recall  f1-score   support

           0       0.95      0.88      0.91     29245
           1       0.41      0.66      0.50      3705

    accuracy                           0.85     32950
   macro avg       0.68      0.77      0.71     32950
weighted avg       0.89      0.85      0.87     32950


% ---------------- XGBoost su dati di test ---------------- %

Parametri ottimali: {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 50}
              precision    recall  f1-score   support

           0       0.94      0.88      0.91      7303
           1       0.38      0.60      0.47       935

    accuracy                           0.84      8238
   macro avg       0.66      0.74      0.69      8238
weighted avg       0.88      0.84      0.86      8238
```

Concentrandoci sui valori di f1-score, notiamo che con scale pos weight siamo riuscito ad avere una performance migliorata del 11% sulla classe minoritaria, mentre sulla classe maggioritaria abbiamo un lieve calo, accettabile in virtù del maggior bilanciamento.
Nonostante quanto detto, il modello resta sbilanciato tra le due classi e non ha risolto i problemi visti con l'undersampling nel xgBoost.

Random forest

Random Forest è un algoritmo di machine learning che si basa sull'idea di utilizzare un insieme di alberi decisionali per fare predizioni. Questo approccio è noto come "bagging", che sta per **Bootstrap Aggregating**. L'idea di base è quella di costruire diversi alberi decisionali in modo casuale, ognuno addestrato su un sottoinsieme casuale dei dati di addestramento. Questo introduce casualità e aiuta a ridurre il rischio di overfitting migliorando la generalizzazione del modello.

Quando si fa una predizione con Random Forest, ogni albero dell'insieme genera una previsione. Nel caso della classificazione, la previsione finale è determinata da un voto a maggioranza tra tutti gli alberi (ovvero la classe che riceve più voti), mentre per la regressione, la previsione finale è spesso la media delle previsioni di tutti gli alberi.

Iperparametri per la grid search:

```
# Parametri da testare con GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 150, 200],
    #massima profondità albero
    'max_depth': [6, 10, 20, 30, 40, 50, 75, 100, 150, 200],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

Random forest su dataset originale

```
Parametri ottimali: {'max_depth': 30, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 50}

% -------------------- Random Forest test -------------------- %

Average Report Test:
Class 0:
        precision: 0.92
        recall: 0.98
        f1-score: 0.95
        support: 1827.00
Class 1:
        precision: 0.61
        recall: 0.30
        f1-score: 0.40
        support: 232.00


% -------------------- Random Forest train -------------------- %

Average Report Train:
Class 0:
        precision: 0.94
        recall: 1.00
        f1-score: 0.97
        support: 34721.00
Class 1:
        precision: 0.95
        recall: 0.52
        f1-score: 0.67
        support: 4408.00
```

osservazioni su random forest sul df originale

- **Overfitting**: il modello sembra soffrire di overfitting. Questo è evidente dal fatto che le metriche di precisione, recall e f1-score sono significativamente più alte sul set di addestramento rispetto al set di test. Ad esempio, la precisione per la classe 1 è del 95% sul set di addestramento, ma scende al 61% sul set di test.
- **Bilanciamento delle classi**: come visto anche in precedenza abbiamo uno squilibrio tra le classi che verrà affrontato successivamente con l'undersampling.
- **Parametri del modello**: la grid search ha scelto una profondità massima di 30 per l'albero, ciò potrebbe star contribuendo all'overfitting, introducendo un'eccessiva complessità e attaccamento ai dati.
- **Metriche di valutazione**: la precisione per la classe 1 è relativamente bassa, il che indica che il modello ha un numero relativamente alto di falsi positivi per questa classe. Inoltre, il recall per la classe 1 è solo del 30%, il che indica che il modello non è in grado di identificare una grande percentuale dei veri positivi per questa classe.

Concludendo, ciò che possiamo evincere è che il random forest sul dataset originale non ha prodotto risultati ottimali, ma sicuramente superiori agli algoritmi xgBoost e adaBoost. Probabilmente ciò è dovuto alla capacità del random forest di introdurre casualità, dovuta al bootstrap Aggregating, spiegato in precedenza.

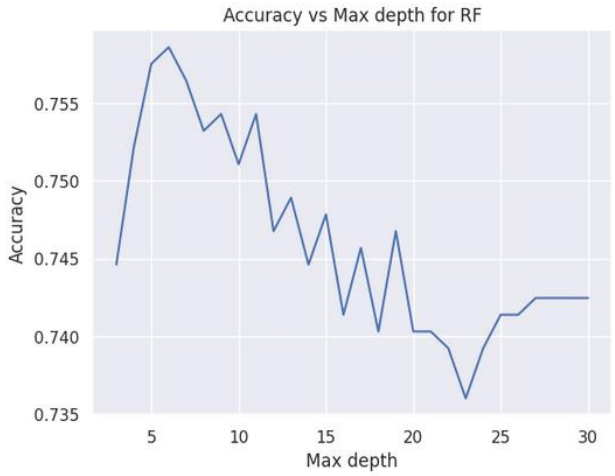
Random forest con undersampling

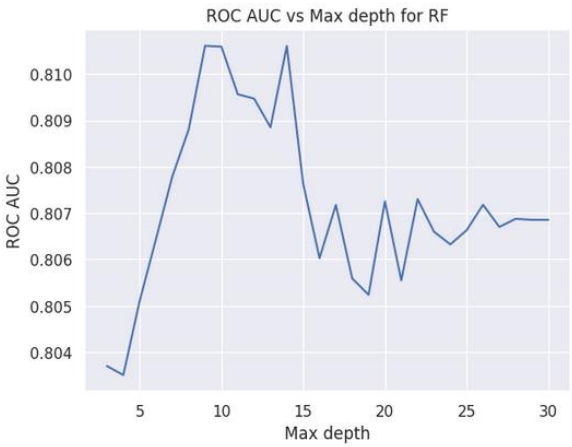
```
Parametri ottimali: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 200}

% -------------------- Random Forest test -------------------- %

Average Report Test:
Class 0:
        precision: 0.71
        recall: 0.86
        f1-score: 0.78
        support: 464.00
Class 1:
        precision: 0.82
        recall: 0.64
        f1-score: 0.72
        support: 464.00


% -------------------- Random Forest train -------------------- %

Average Report Train:
Class 0:
        precision: 0.72
        recall: 0.89
        f1-score: 0.79
        support: 4176.00
Class 1:
        precision: 0.85
        recall: 0.65
        f1-score: 0.74
        support: 4176.00
```

Possiamo notare una diminuzione della maxDepth a 10, rispetto al modello su dataset originale dove maxDepth = 30. In effetti, ciò era prevedibile, con un calo della mole d'esempi ci aspettiamo una minor maxDepth necessaria per raggiungere un elevato score di accuracy.

Ulteriori osservazioni:

- **Bilanciamento delle classi**: L'undersampling sembra aver aiutato a bilanciare le classi nel set di dati.
- **Miglioramento delle prestazioni**: Le metriche di precisione, recall e f1-score per la classe 1 sono migliorate nel set di test rispetto ai risultati precedenti. Questo indica che il modello è ora in grado di gestire meglio la classe 1 (classe minoritaria).
- **Riduzione dell'overfitting**: Le metriche di precisione, recall e f1-score sono ora più simili tra il set di addestramento e il set di test, il che suggerisce che l'overfitting potrebbe essere stato risolto.
- **Parametri del modello**: I parametri ottimali trovati indicano che il modello Random Forest è meno complesso rispetto al modello precedente, con una profondità massima di 10. Questo potrebbe aver contribuito a ridurre l'overfitting.





Possiamo notare dai due grafici come ci sia una forte fluttuazione inerente alla maxDepth, sia per la roc che per l'accuracy. Ciò denota una forte sensibilità del modello rispetto la profondità.

MLP – Multi Layer Perceptron

Ora, procediamo a testare un modello basato su rete neurale, applicando sempre k fold cross validation e grid search, la quale potrà scegliere tra i seguenti parametri:

```
# Parametri da testare con GridSearchCV
param_grid = {
    'hidden_layer_sizes': [(100,), (50, 50), (50, 100, 50)],
    'activation': ['relu', 'tanh', 'logistic'],
    'alpha': [0.0001, 0.05],
}
```

MLP su dataset originale:

```
Parametri ottimali: {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100,)}

% -------------------- MLP test -------------------- %

Average Report Test:
Class 0:
        precision: 0.91
        recall: 0.98
        f1-score: 0.94
        support: 1827.00
Class 1:
        precision: 0.61
        recall: 0.22
        f1-score: 0.33
        support: 232.00


% -------------------- MLP train -------------------- %

Average Report Train:
Class 0:
        precision: 0.91
        recall: 0.99
        f1-score: 0.94
        support: 34721.00
Class 1:
        precision: 0.64
        recall: 0.21
        f1-score: 0.31
        support: 4408.00
```

osservazioni sul MLP con df originale

- overfitting: il modello non presenta trade off tra valori delle metriche nel train e nel test, di conseguenza non possiamo notare overfitting da questi risultati.
- performance: come visto finora, vi è differenza di performance tra la classe maggioritaria e minoritaria, nella classe 0 otteniamo ottimi risultati, mentre nella classe 1 i risultati non sono buoni (seppur vi è una discreta precision).

MLP con undersampling

```
Parametri ottimali: {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50)}

% -------------------- MLP test -------------------- %

Average Report Test:
Class 0:
        precision: 0.65
        recall: 0.92
        f1-score: 0.76
        support: 464.00
Class 1:
        precision: 0.86
        recall: 0.51
        f1-score: 0.64
        support: 464.00


% -------------------- MLP train -------------------- %

Average Report Train:
Class 0:
        precision: 0.64
        recall: 0.92
        f1-score: 0.76
        support: 4176.00
Class 1:
        precision: 0.86
        recall: 0.49
        f1-score: 0.63
        support: 4176.00
```

osservazione su MLP con undersampling rispetto la versione su df originale:

- per la classe 0(maggioritaria): possiamo notare un generale calo delle performance delle metriche, con un guadagno in termini di credibilità però dei dati ottenuti.
- per la classe 1 (minoritaria): possiamo notare un ottimo miglioramento delle performance, con la classe minoritaria che ha beneficiata dal bilanciamento delle due classi

Concludendo per l'MLP non possiamo notare la presenza di attaccamento ai dati.

Apprendimento probabilistico

L'apprendimento probabilistico si basa sulla teoria delle probabilità per costruire modelli che rappresentano le relazioni tra variabili. Tali modelli descrivono l'incertezza nei dati e consentono di fare previsioni informate considerando la probabilità degli eventi.

Nel contesto di questo capitolo, ci concentreremo sull'utilizzo del modello probabilistico Naive Bayes. Il Naive Bayes è un tipo di algoritmo di classificazione probabilistica che si basa sul teorema di Bayes. Esso assume l'indipendenza condizionale tra le caratteristiche, noto come "assunzione naive". In altre parole, l'algoritmo presuppone che le caratteristiche siano indipendenti tra loro dato il valore della variabile di classe. Questa semplificazione può essere efficace in molte situazioni, sebbene sia un'assunzione semplificatrice.

Naive Bayes

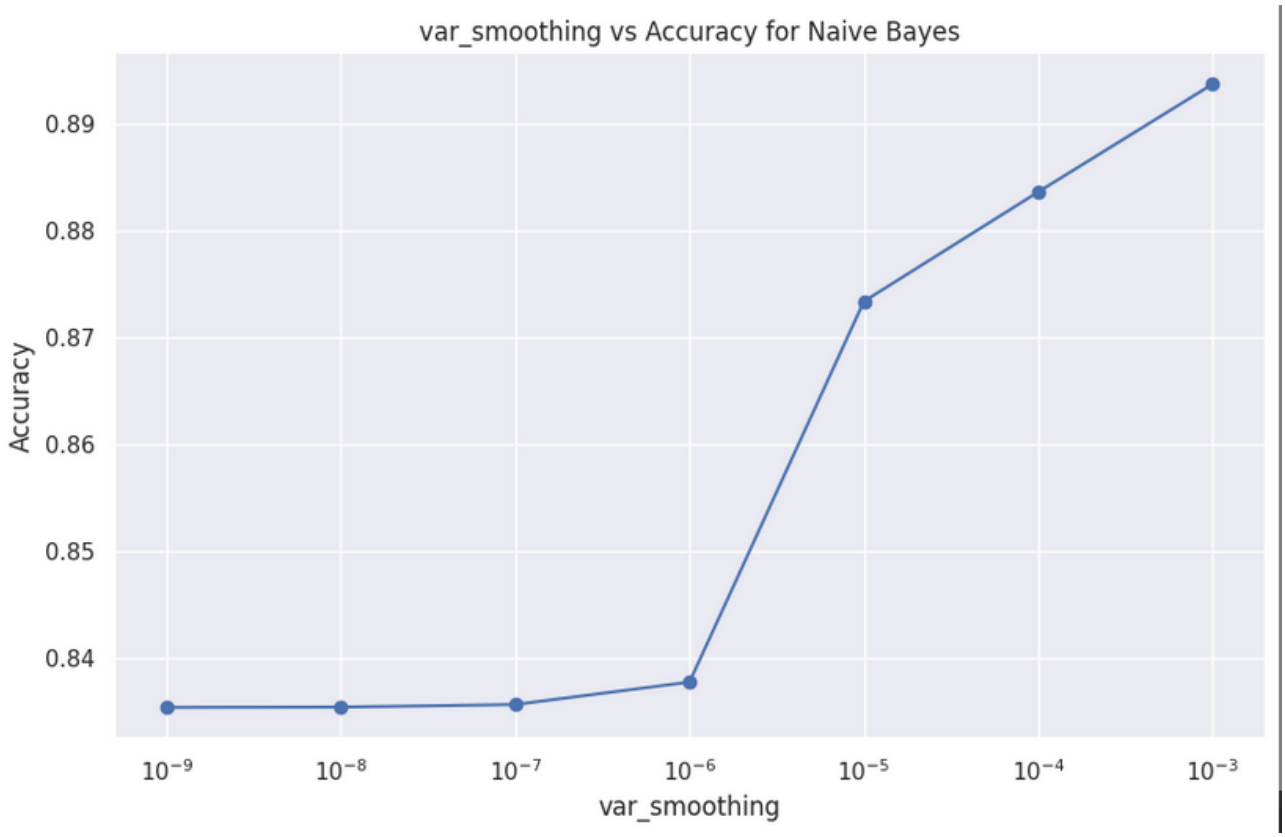
Come fatto per gli altri modelli si applicheranno grid search e k fold cross validation.

Risultati:

```
% ---------------- Naive Bayes su dati di test ---------------- %

Parametri ottimali: {'var_smoothing': 0.001}
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      7303
           1       0.52      0.26      0.35       935

    accuracy                           0.89      8238
   macro avg       0.72      0.62      0.64      8238
weighted avg       0.87      0.89      0.87      8238


% ---------------- Naive Bayes su dati di training ---------------- %

Parametri ottimali: {'var_smoothing': 0.001}
              precision    recall  f1-score   support

           0       0.91      0.97      0.94     29245
           1       0.55      0.28      0.37      3705

    accuracy                           0.89     32950
   macro avg       0.73      0.62      0.66     32950
weighted avg       0.87      0.89      0.88     32950
```

Osservazioni:

- **Precisione**: La precisione è alta per la classe 0 (0.91 per i dati di test e training), ma significativamente più bassa per la classe 1 (0.52 per i dati di test e 0.55 per i dati di training). Questo indica che il modello è più preciso nel prevedere la classe 0 rispetto alla classe 1.
- **Recall**: Il recall per la classe 0 è molto alto (0.97 per i dati di test e training), mentre per la classe 1 è piuttosto basso (0.26 per i dati di test e 0.28 per i dati di training). Questo suggerisce che il modello è in grado di identificare correttamente la maggior parte dei veri positivi per la classe 0, ma non per la classe 1.
- **F1-Score**: L'F1-score per la classe 0 è alto (0.94 per i dati di test e training), mentre per la classe 1 è basso (0.35 per i dati di test e 0.37 per i dati di training). L'F1-score è una misura che combina precisione e recall; quindi, questi risultati indicano che il modello ha una performance complessivamente migliore per la classe 0 rispetto alla classe 1.
- **Accuracy**: L'accuracy è la stessa per i dati di test e di training (0.89), il che indica che il modello ha una buona capacità di prevedere correttamente sia i positivi che i negativi.



Osservando inoltre il grafico, la grid search ha scelto il punto di picco del var_smoothing, che in questo caso non risulta portare ad overfitting.

Conclusioni

Non sempre l'undersampling ha portato ai risultati sperati. Nel caso del K-NN, il miglioramento ottenuto è stato degno di nota, con un maggiore equilibrio tra le classi, seppur diminuendo le performance della classe maggioritaria che però guadagna di credibilità. Lo stesso non si può dire dell'adaBoost, che ha offerto performance incoraggianti, seppur con il solito sbilanciamento, per poi andare in overfitting con l'undersampling. Anche l'xgBoost dopo l'undersampling ha dato leggeri segnali di overfitting nella classe minoritaria. L'ipotesi è che il dataset in uso, non risponda bene con l'aumento di complessità dei modelli, di fatti, il K-NN ha offerto ottime prestazioni.

Il Random Forest ha dato risultati apprezzabili, ciò ritengo sia dovuto alla sua capacità di introdurre randomicità attraverso il Bootstrap Aggregating.

Discreti risultati per l'MLP, il quale però introduce complessità e ricade nelle ipotesi precedenti.
Non particolarmente degni di nota i risultati del naive Bayes, che mantiene il solito pattern di ottimi risultati sulla classe maggioritaria contro scarsi risultati nella minoritaria (con assenza di overfitting).

Possibili sviluppi futuri

In futuro si potrebbe effettuare maggior sperimentazione sui parametri utilizzati dalla grid search. Sicuramente con maggiore esperienza e risorse si possono capire come ottenere migliori performance sui modelli in uso.

Si potrebbe inoltre testare le performance di un SVM che inizialmente era stato preso in considerazione, ma durante lo sviluppo del progetto ci si è resi conto che in assenza di risorse adeguate, i tempi sono lunghi e si è deciso di accantonarlo. L'SVM in particolare, per l'addestramento richiede risorse discretamente potenti.

In generale, si è dovuto affrontare la necessità di testare varie idee e tentativi con le risorse offerte da google colab, cercando un equilibrio tra le risorse.