RFD FEATURE SELECTION FOR FAKE ACCOUNT CLASSIFICATION

Corso di Fondamenti di Data Science e Machine Learning A.A. 2021/2022

Prof. Giuseppe Polese *Gianluca Spinelli matr.: 0522500981*

Sommario

1. Introduzione

I social network consentono di condividere informazioni tra utenti di ogni età, in ogni momento e in ogni parte del mondo. Le piattaforme di interazione sociale come Instagram, Twitter, Tumblr, ecc. hanno un impatto significativo sulla vita quotidiana dei loro utenti e dell'intera società. Il tipico profilo di un utente di un social network può contenere diversi indizi relativi a emozioni, passioni, interessi e caratteristiche del proprietario del profilo. Per questi motivi, molte persone tendono ad avere una vita virtuale che non `e necessariamente riconducibile a quella reale. Tuttavia, in una società in cui spesso le amicizie si rafforzano attraverso Facebook, le emozioni si diffondono attraverso Instagram, il lavoro si trova attraverso LinkedIn e così via, `e necessario definire meccanismi di controllo in grado di prevenire l'uso improprio di tali piattaforme di condivisione, individuando automaticamente eventuali utenti malintenzionati, ovvero spammer, bot, profili falsi, ecc. Un aspetto fondamentale da monitorare in un social network `e la popolarità di un profilo, testimoniata dal numero di follower. Questa popolarità può essere aumentata tramite l'acquisto di follower falsi ad un prezzo basso. Questa pratica potrebbe essere pericolosa in quanto potrebbe essere usata per acquisire popolarità per promuovere particolari servizi, siti ingannevoli, servizi truffa ecc.

È necessario quindi identificare i profili falsi presenti sui vari social network. Per questo progetto si utilizzerà un dataset di profili di Twitter contenente per ogni account 15 caratteristiche.

In un precedente lavoro queste caratteristiche sono state aumentate con dei metadati estratti dal dataset, ovvero le Dipendenze Funzionali Rilassate.

1.1. Data Preprocessing

La Data Preprocessing è una fase che prende dati grezzi e li trasforma in un formato che può essere compreso e analizzato da computer. Spesso si sente la frase "garbage in, garbage out". Ciò significa che se usiamo dati errati "sporchi" per addestrare il modello, otterremo un modello cattivo. Avere dei dati preelaborati è ancor più importante di avere degli algoritmi potenti, al punto che i modelli di machine learning addestrati con dati errati potrebbero effettivamente essere dannosi per l'analisi che si sta cercando di fare, dandoti risultati "spazzatura".

Nell'ambito della Data Cleaning, in questo progetto viene affrontato il concetto di Dipendenza Funzionale Rilassata nella fase di Feature Selection; ovvero si vuole testare alcune tecniche di Feature Selection note in letteratura su un insieme di Dipendenze Funzionali Rilassate estrapolate per capire se queste tecniche possono essere applicate e che risultati si ottengono.

Non è necessario utilizzare tutte le feature, ovvero gli attributi predittori utili per la stima, ma dovremmo utilizzare solo quelle feature che sono veramente importanti. Il processo di selezione delle sole feature importanti per la costruzione del modello viene chiamato Feature Selection ed applicarlo porta numerosi vantaggi, quali:

1) Semplifica il modello: si hanno meno dati, quindi una occupazione di memoria minore ed una riduzione della complessità che facilità l'interpretazione;
2) Riduce il tempo di addestramento;

3) Riduce l'overfitting (ovvero il modello di comporta troppo bene sui dati di addestramento mentre si comporta male sui dati utilizzati per testarlo)
4) Migliora l'accuratezza del modello

Sull'ultimo vantaggio bisogna fare un appunto: anche se dopo aver applicato il processo di Feature Selection, le metriche di performance del modello dovessero rimanere invariate o al più diminuire di poco (10%), non bisognerebbe considerare ciò come un risultato "negativo" del processo di selezione delle feature perché, pur avendo una piccola diminuzione delle performance, si ha una riduzione del numero di predittori (numero di colonne del dataset) che comporta una diminuzione del tempo di addestramento e della complessità del modello, aspetti fondamentali nel mondo del Machine Learning.

2. Richiami di RFD e Feature Selection

2.1. RFD

Nel tempo è nata la necessità di tecniche non solo per reperire i dati ma anche per estrarre informazioni e conoscenza da essi, soprattutto nell'ambito dei Big Data.

Quindi abbiamo la necessità di applicare tecniche di Data Profiling per capire se all'interno dei dati ci sono dei Pattern, ovvero ci sono dei dati che hanno una certa struttura che si ripete. Tra le varie domande che ci poniamo quando cerchiamo questi pattern ci chiediamo anche se esistano delle Dipendenze funzionali.

Le Dipendenze Funzionali sono un particolare tipo di Metadato (dato che non vediamo all'interno dell'insieme di dati ma che necessita di essere estratto) che determina la relazione di un attributo con un altro attributo all'interno di un dataset. Esiste una Dipendenza Funzionale tra due sottoinsiemi di attributi X e Y se per ogni coppia di tuple t_1 t_2 aventi gli stessi valori su X, t_1 e t_2 hanno gli stessi valori anche sugli attributi Y.

Non sempre però il valore delle tuple sugli attributi della parte sinistra è precisamente uguale ma posso essere simili. Di recente c'è stato un rinnovato interesse per le dipendenze funzionali dovuto alla possibilità di usarle in diversi domini come: Data Cleaning, Query Relaxation, Record Matching.

Tuttavia, c'è stata la necessità di rilassare alcuni vincoli della definizione di dipendenza funzionale standard.

Sono state introdotte le RFD che ammettono la possibilità che ci siano rilassamenti o approssimazioni: Dipendenze Funzionali Rilassate.

Quindi il concetto di Dipendenza Funzionale può essere modificato in: "*Se 2 tuple sono simili su A, sono simili su B*".

Esistono due criteri di rilassamento: sul Confronto e sul grado di soddisfacibilità.

Rilassare una dipendenza funzionale sul confronto vuol dire ammettere che due tuple su un attributo X abbiamo valori simili e non uguali.

Rilassare sul grado di soddisfacibilità significa invece ammettere la possibilità che la dipendenza valga su un sottoinsieme di tuple e non su tutto il dataset.

Queste dipendenze funzionali rilassate risultano essere molto importanti per scopi di Query Relaxation, Data Cleaning e Record Matching. In particolare, i vincoli definiti per le dipendenze

funzionali canoniche essendo allentati, permettono di catturare incongruenze nei dati reali, modelli di dati semanticamente correlati o relazioni semantiche in tipi di dati complessi.

2.2. Feature Selection

Non è detto che aumentando il numero di features, l'accuratezza aumenti. Esiste una soglia k, oltre la quali l'accuracy non aumenterà all'aumentare delle features. Dopo questa soglia k, l'accuracy inizia a decrescere con l'aumentare del numero di features: ciò è conosciuto col nome di Curse of Dimensionality.

La Curse of Dimensionality può essere risolta con:

1) PCA, ovvero Dimensionality Reduction, che va a modificare o trasformare feature in termini di dimensione;
2) Feature Selection, che esclude alcune feature senza modificarle.

In base al tipo di dati che si ha a disposizione si possono distinguere due tipi di tecniche di Feature Selection:

1) Supervised, usabili per labeled data, ovvero quando abbiamo a che fare con addestramenti supervisionati;
2) Unsupervised, usabili per non labeled data, ovvero quando abbiamo a che fare con addestramenti non supervisionati.

Nel nostro caso abbiamo a che fare in labeled data, quindi le tecniche disponibili in letteratura sono le seguenti:

1) Filter Methods
2) Wrapper Methods
3) Embedded Methods

Filter Methods

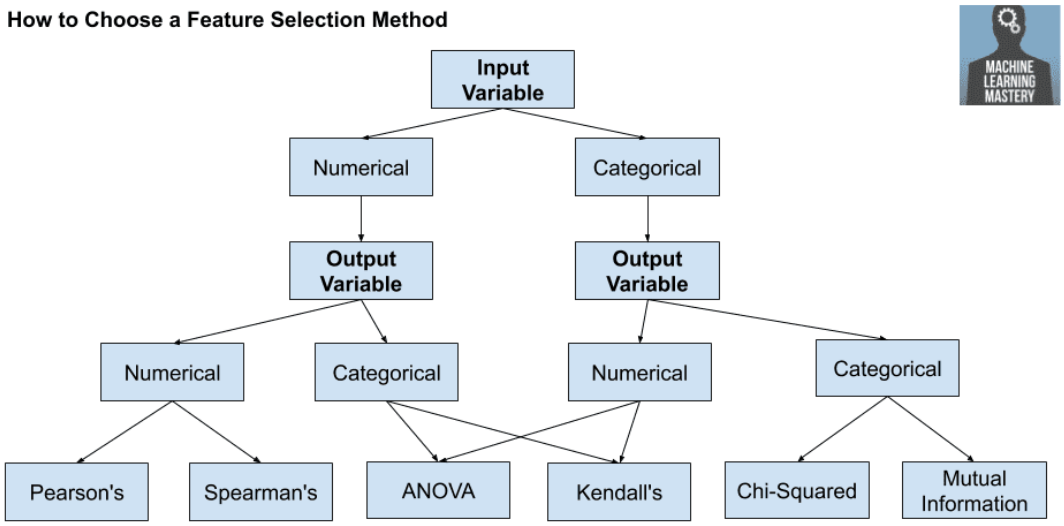
I Filter Methods sono metodi indipendenti dal tipo di algoritmo di Machine Learning che si utilizza Le funzionalità selezionate utilizzando i metodi di filtro possono essere utilizzate come input per qualsiasi modello di machine learning. Un altro vantaggio dei metodi di filtraggio è che sono molto veloci. I metodi di filtro sono generalmente il primo passaggio in qualsiasi pipeline di selezione delle funzionalità. Questi metodi possono essere classificati in due categorie: metodi di filtro **univariati** e **multivariati**:

I metodi di filtro **univariati** sono dei metodi in cui le singole caratteristiche sono classificate in base a criteri specifici, come ad esempio la varianza della feature. Uno dei principali svantaggi è che possono selezionare funzionalità ridondanti perché la relazione tra le singole funzionalità non viene presa in considerazione durante le decisioni.

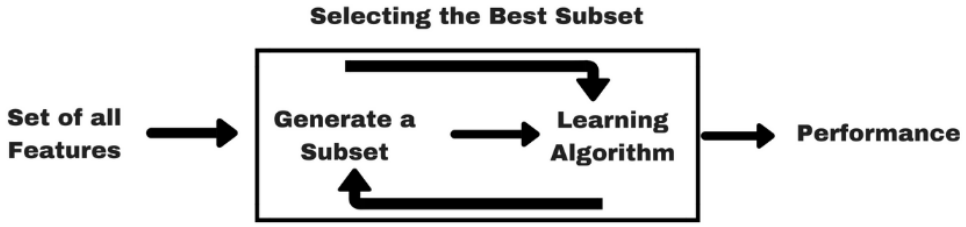
I metodi di filtro **multivariati** sono in grado di rimuovere le funzionalità ridondanti dai dati poiché tengono conto della relazione reciproca tra le funzionalità, calcolata mediante test statistici. I metodi di questo tipo possono essere utilizzati per rimuovere feature correlate.

In base al tipi di feature e di output che si hanno si possono utilizzare relativi test statistici.

Ad esempio, nel nostro caso abbiamo a che fare con Variabili Input (Feature) numeriche e output numerici quindi è possibile utilizzare l'indice di correlazione Pearson's. Nello schema seguente è possibile visionare chiaramente tutti i possibili metodi statistici applicabili.



Wrapper Methods



Nei metodi wrapper, proviamo a utilizzare un sottoinsieme di funzionalità e ad addestrare un modello che le utilizza. Sulla base delle inferenze che otteniamo dal modello precedente, decidiamo di aggiungere o rimuovere funzionalità dal tuo sottoinsieme. Il problema si riduce essenzialmente a un problema di ricerca dove selezionano quelle funzionalità che si traducono nel modello con le prestazioni migliori. Questi metodi sono generalmente molto costosi dal punto di vista computazionale.

Alcuni esempi comuni di metodi wrapper sono:

- **Forward Selection:** la selezione in avanti è un metodo iterativo in cui iniziamo senza avere alcuna caratteristica nel modello. In ogni iterazione, continuiamo ad aggiungere la caratteristica che meglio migliora il nostro modello fino a quando l'aggiunta di una nuova variabile non migliora le prestazioni del modello.
- **Backward Selection:** nell'eliminazione all'indietro, iniziamo con tutte le funzionalità e rimuoviamo la funzionalità meno significativa ad ogni iterazione che migliora le prestazioni del modello. Lo ripetiamo fino a quando non si osserva alcun miglioramento sulla rimozione delle funzionalità.
- **RFE (Recursive Feature Elimination):** è un algoritmo di ottimizzazione greedy (avido) che mira a trovare il sottoinsieme di funzionalità con le migliori prestazioni. Crea ripetutamente modelli e tiene da parte la funzionalità migliore o peggiore ad ogni

iterazione. Costruisce il modello successivo con le feature rimaste fino a quando tutte le feature non sono esaurite. Quindi classifica le caratteristiche in base all'ordine della loro eliminazione.

Uno dei modi migliori per implementare la selezione delle caratteristiche con i metodi wrapper consiste nell'usare il pacchetto Boruta che trova l'importanza di una caratteristica creando caratteristiche ombra.

Esso funziona nel seguente modo:

1. Questo metodo prima crea casualità nelle Feature creando feature duplicate e mescolando i valori in ogni colonna. Queste features sono chiamate Funzionalità ombra.

2. Successivamente addestra un classificatore (Random Forest) sul dataset e calcola l'importan za delle features usando Mean Decrease Accuracy oppure Mean Decrease Gini Impurity.

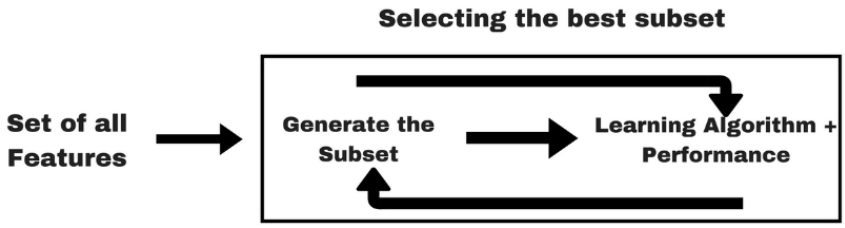
3. Successivamente, l'algoritmo controlla ciascuna delle caratteristiche reali se ha un'importan za maggiore. Cioè, se la feature ha un punteggio Z più alto del punteggio Z massimo delle sue f eature d'ombra.

4. Ad ogni iterazione, l'algoritmo confronza lo Z score delle copie ombra delle features con le fe ature originali per vedere se quest'ultime performano meglio dele prime. Se si, l'algoritmo mar cherà questa feature come importante.

Essengo costruito con RandomForest Boruta effettua una selezione delle feature migliore con modelli basati su alberi come RF o XGBoost, ma risulta essere un valido metodo di Feature Sele ction anche per modelli di classificazione come LogisticRegression o SVM.

Una volta ottenuto l'insieme di feature selezionate ovviamente questo può essere applicato a qualsiasi tipo di modello.

Embedded Methods



I metodi embedded combinano le qualita' dei metodi filter e wrapper. È implementato da algoritmi che hanno i propri metodi di selezione delle funzionalità incorporati.

Alcuni degli esempi più popolari di questi metodi sono la regressione LASSO e RIDGE che hanno funzioni di penalizzazione integrate per ridurre l'overfitting.

È meno costoso dal punto di vista computazionale del metodo wrapper e meno soggetto a overfitting.

I metodi selezionati per questo esperimento sono:

- Variance Filtering Method
- Correlation Based Filtering Method
- Boruta Method

3. Scopo del progetto

Lo scopo di questo progetto è sperimentare tecniche di Feature Selection note sull'insieme di Dipendenze Funzionali Rilassate (RFD) per verificarne la validità.

L'estrapolazione di RFD a partire da un dataset porta ad un cospicuo aumento delle colonne all'interno del dataset e quindi delle Feature, ma non è detto che tutte queste nuove feature contribuiscano alla costruzione di un modello di machine learning che dia migliori risultati.

Per tal motivo, lo scopo è quello di determinare tramite tecniche di Feature Selection, quali delle feature calcolate tramite RFD, in aggiunta alle 15 feature di base, sono importanti nel processo di addestramento di un modello di Machine Learning. Inoltre, si vuole ridurre al minimo il numero di RFD cercando di preservare le performance dei rispettivi modelli addestrati su tutte le RFD estratte.

4. Dataset

Sono stati presi in considerazione 9019 account, ciascuno categorizzato con una delle seguenti classi: Real(1) e Fake(0). Nella tabella successiva vengono illustrate le 16 feature caratterizzanti ciascun account. Il dataset è composto da 5706 account Real e 3313 account Fake.

#	Feature	Descrizione
1	name	Nome scelto dal proprietario dell'account.
2	screen_name	Identificatore associato al proprietario dell'account.
3	location	Posizione indicata dal proprietario dell'account.
4	url	Valore booleano che rappresenta l'assenza o meno dell'URL impostato dal proprietario dell'account.
5	description	Valore booleano che rappresenta l'assenza o meno della descrizione impostata dal proprietario dell'account.
6	followers_count	Numero di follower attuali associati al proprietario dell'account.
7	friends_count	Numero di utenti seguiti dal proprietario dell'account.
8	listed_count	Numero di elenchi pubblici in cui compare il proprietario dell'account.
9	created_at	Data e ora in cui è stato creato l'account.
10	favourites_count	Numero di tweet generati dal proprietario dell'account.
11	geo_enabled	Valore booleano che rappresenta il geottaging dei tweet relativi al proprietario dell'account.
12	statuses_count	Numero di retweet eseguiti dal proprietario dell'account.
13	lang	Codice associato alla lingua specificata dal proprietario dell'account.
14	default_profile	Valore booleano che indica le modifiche relative al tema o allo sfondo del proprietario dell'account.
15	default_profile_image	Valore booleano che rivela se l'immagine predefinita di Twitter è stata modificata dal proprietario dell'account.
16	class	Classe di appartenenza dell'account.

5. Primo step: addestramento di tre modelli di classificazione

In questa prima fase sono stati implementati tre modelli di classificazione binaria, diversi dai cinque già presenti nel notebook. Ciascuno dei tre modelli è stato addestrato e testato in primis sulla baseline, quindi il dataset "dataset.csv" comprendente le sole 15 colonne e successivamente su tutti i dataset aumentati con le Dipendenze Funzionali Rilassate estratte.

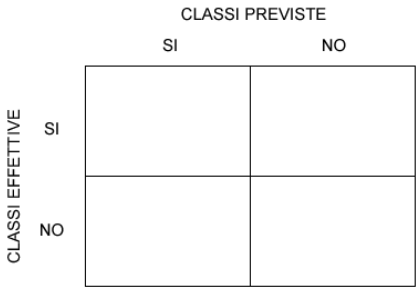
L'obiettivo di questo primo step è quello di verificare che, anche con questi tre modelli di classificazione, l'aggiunta di Dipendenze Funzionali Rilassate al dataset di baseline comporta un miglioramento delle performance rispetto al modello addestrato sul dataset baseline.

5.1. Valutazione delle performance

Per valutare ciascuno dei tre modelli sono state utilizzare varie metriche per accettarsi delle prestazioni di questi. In particolare, per ogni modello è stata riportata in primis l'Accuracy e successivamente è stata generata la Confusion Matrix con il classification report per analizzare Precision, Recall, f1-score, macro svg, weighted avg; è stata generata la ROC Curve e nel caso dell'implementazione con MLP è stato plottato l'andamento di training e testing nelle varie epoche ed è stata generato il grafico della Loss Curve.

La confusion matrix è particolarmente importante perché nell'ambito di un problema di classificazione binaria come il nostro, permette di ottenete tutte le principali metriche di un modello di Machine Learning.

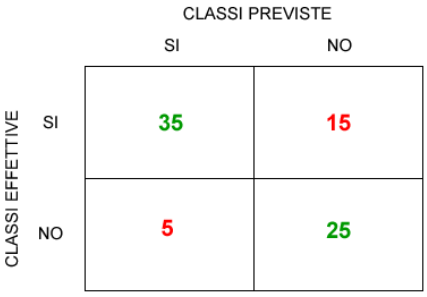
La matrice è formata da due righe e due colonne così strutturata:



- Sulle righe troviamo le classi effettive, cioè le classi delle risposte corrette.
- Sulle colonne indichiamo le classi di previsione, cioè le classi delle risposte del modello.

All'interno delle caselle vengono caricate le risposte del modello alle varie istanze di testing.

Ad esempio, un modello analizza 80 e-mail e deve classificarle come spam/no spam. In 60 casi il modello classifica correttamente mentre in 20 sbaglia.



5.2. Hyperparameters tuning

È raro che un modello funzioni al livello desiderato al primo tentativo. Per trovare la giusta soluzione ad un problema di classificazione o regressione, spesso bisogna passare attraverso un ciclo iterativo . Ci sono più pezzi che si uniscono per risolvere il puzzle di apprendimento automatico. Potrebbe essere necessario addestrare e valutare più modelli che includono impostazioni e algoritmi di dati diversi, eseguire operazioni di Feature Engineering diverse volte o persino aumentare i dati. Questo ciclo comporta anche la modifica degli **iperparametri** del modello.

Gli iperparametri sono le manopole o le impostazioni che possono essere regolate prima di eseguire un processo di addestramento per controllare il comportamento di un algoritmo di Machine Learning. Possono avere un grande impatto sull'addestramento del modello in quanto riguarda il tempo di addestramento, i requisiti delle risorse infrastrutturali (e di conseguenza il costo), la convergenza e l'accuratezza del modello.

I parametri del modello vengono appresi come parte del processo di training, mentre i valori degli iperparametri vengono impostati prima dell'esecuzione del processo di training e non cambiano durante il training.

Nel nostro caso i valori da assegnare agli iperparametri del modello verranno appresi mediante un algoritmo di ricerca detto GridSearch.

Questo algoritmo imposta una griglia composta da iperparametri e dai loro diversi valori. Per ogni possibile combinazione viene addestrato un modello e viene prodotto un punteggio sui dati di validazione. Con questo approccio, viene provata ogni singola combinazione dei possibili valori di iperparametro forniti. Sebbene l'approccio esegua un'analisi approfondita di tutte le possibili combinazioni, può essere molto inefficiente in termini di tempo e costi di formazione.

5.3. Modelli implementati

I tre modelli scelti per il task di classificazione binaria sono:

- kNN Classifier
- MLP Multilayer Perceptron Classifier
- AdaBoost Classifier

Per ogni modello di classificazione è stato effettuato uno split dell'intero dataset in training/test set dell'80-20%.

Ciò significa che il nostro modello verrà addestrato sull'80% del dataset totale ed una terminato l'addestramento, il modello verrà testato sul 20% restante dei dati restanti.

Si precisa che sono state mantenute le trasformazioni applicate sui dati nei cinque modelli già implementati, ovvero la trasformazione del campo 'crated_at' nel formato "Mon – YYYY" e la conversione dei campi testuali 'name','screen_name','location','created_at','lang' in campi numerici.

Queste trasformazioni ci permettono di ottenere tutte variabili numeriche. Questa è la procedura standard per ridimensionare i nostri dati durante la creazione di un modello di apprendimento automatico in modo che il nostro modello non sia sbilanciato verso una

caratteristica particolare del dataset e allo stesso tempo impedisca al nostro modello di apprendere le caratteristiche/valori/trend dei nostri dati di test.

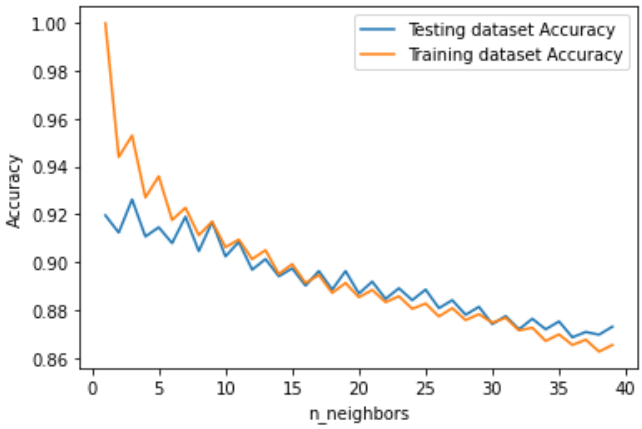
kNN

Il primo modello implementato è kNN. L'algoritmo k-nearest neighbors è un classificatore di apprendimento supervisionato che utilizza la prossimità per effettuare classificazioni o previsioni sul raggruppamento di un singolo dato. Viene generalmente utilizzato come algoritmo di classificazione in quanto si basa sul presupposto che punti simili possono essere trovati l'uno vicino all'altro.

L'input è costituito dai *k* esempi di addestramento più vicini nel dataset. Un oggetto è classificato in base a un voto di pluralità dei suoi vicini, con l'oggetto assegnato alla classe più comune tra i suoi *k* vicini più prossimi (*k è un* intero positivo , tipicamente piccolo). Se *k* = 1, allora l'oggetto viene semplicemente assegnato alla classe di quel singolo vicino più vicino.

Step 1: Costruzione Baseline con k = 1 - 40

Come primo step è stato costruito un classificatore iterando il valore di k, cioè il valore del numero nel range da 1 a 40 e per ognuno di questi ne è stata misurata l'accuracy. A termine addestramento è stato generato un grafico che mostra i vari livelli di accuracy su training e test set per i diversi k.



Come si può notato il grafico identifica che il miglior valore di k dovrebbe essere pari ad 1 (accuracy 0.9182).

Step 2: Tuning Parametri

Considerando che sul modello costruito non è stato fatto alcun tuning dei parametri come la distanza ed il tipo di pesi e valutando questi risultati poco credibili il lavoro è proseguito con l'addestramento degli iperparametri mediante GridSearch.

Durante questa fase è stato definito un dizionario di tutti i valori che vogliamo testare per ottenere il miglior modello da addestrare.

'n_neighbors' rappresenta il valore di k (il numero dei vicini da considerare);

'weights' rappresenta la funzione di peso utilizzata per la previsione. Questa nel caso assuma valore 'uniform' considera tutti i punti in ogni vicinato ponderatamente uguali invece, con valore 'distance', i punti più vicino ad un punto di interrogazione avranno un'influenza maggiore rispetto ai vicino che sono più lontani.

'p' rappresenta la misura di distanza. Con p=1 il classificatore utilizzerà la distanza di Manhattan, mentre con p=2 il classificatore utilizzerà la distranza Euclidea.

Step 3: Training

Successivamente all'addestramento sono stati stampati i best_params_ che l'algoritmo GridSearch ha individuato ed è stato addestrato il modello.

Step 4: Testing e Valutazione

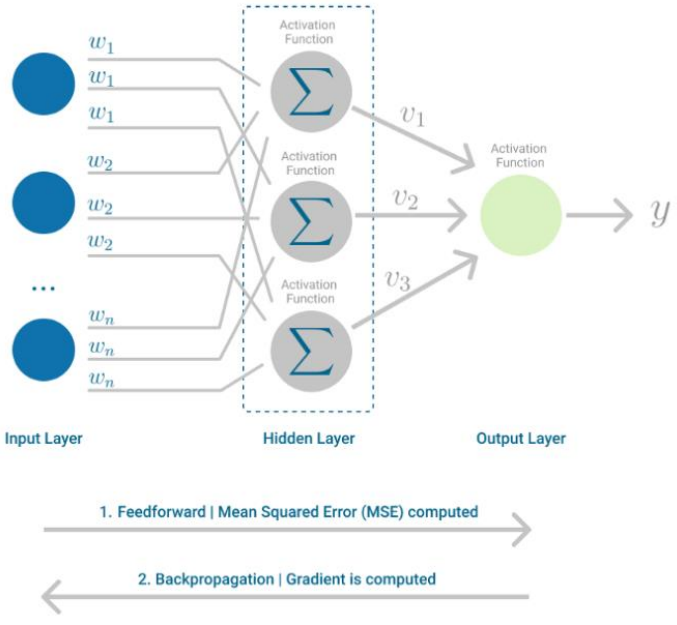
Dopo aver addestrato il modello questo è stato testato sui dati di testing ('x_test') e sono state misurate le varie metriche di performance.

Qui di seguito possiamo vedere un esempio di confusion matrix.

Per ogni addestramento eseguito sui dataset aumentati con le dipendenze funzionali rilassate sono stati ottenute migliori prestazioni rispetto alla baseline.

MLP Multilayer Perception Classifier

Il secondo modello di classificazione binaria implementato utilizza una rete neurale multilivello. Questo tipo di rete neurale è composta da strati densi completamente collegati, che trasformano qualsiasi dimensione di input nella dimensione desiderata. Un perceptron multistrato ha un livello di input con un nodo per ogni input, ha un livello di output con un singolo nogo per ogni output e può avere un numero qualsiasi di hidden layers, cioè livelli intermedi, con un numero qualsiasi di nodi. Inoltre, mentre nel Perceptron il neurone deve avere una funzione di attivazione che imponga una soglia, come ReLU o sigmoide, i neuroni in un Perceptron multistrato possono utilizzare qualsiasi funzione di attivazione arbitraria.



L'implementazione di questa rete neurale è avvenuta tramite la libreria Keras.

Step 1: Costruzione rete Baseline

Nel primo step della realizzazione della rete si sono effettuati diversi addestramenti per trovare una giusta combinazione tra numero di livelli e numero di nodi per livello. Dopo aver

seguito vari approcci proposti in letteratura, la realizzazione della rete è avvenuta con la seguente logica:

- Il livello di input è un livello Denso con numero di nodi pari alla dimensione dell'input e funzione di attivazione 'relu'
- Due hidden layer con funzione di attivazione 'relu' e numero di nodi pari al numero totale delle feature presenti nel dataset
- Livello di output con un solo nodo e funzione di attivazione 'sigmoid'
- Ottimizzatore 'Adam' impostato con un learning_rate = 0.005 ed un epsilon = 1e-08

Step 2: Training della rete

Una volta costruito e compilata la rete è stato effettuato l'addestramento specificando un validation_split = 0.2, un numero di epoche = 100 ed una batch size = 100:

Per ogni epoca dell'addestramento si sono visionati i valori di loss, accuracy, val_loss e val_accuracy, concentrandoci maggiormente sui valori val_# in quanto rappresentano i valori di perdita ed accuratezza ottenuti sul set di validazione a termine di ogni epoca.

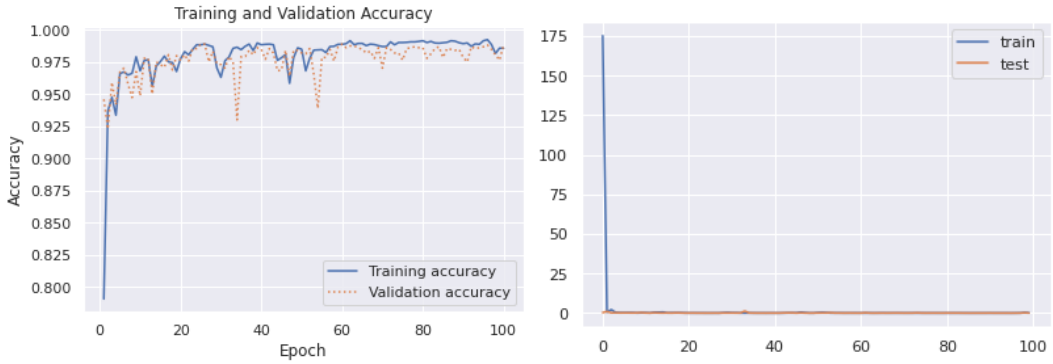
Per valutare le prestazioni di un modello, utilizziamo una misura nota come loss. In particolare**, questa perdita quantifica l'errore prodotto dal modello**. Un valore di perdita elevato di solito significa che il modello sta producendo un output errato, mentre un valore di perdita basso indica che ci sono meno errori nel modello.

Step 3: Valutazione performance

La valutazione delle performance è avvenuta in più passaggi:

Come primo step si sono visualizzati i valori di best accuracy e best loss e si è osservato a che punto della storia è stato raggiunto il valore più alto di accuracy. Questo passaggio è stato effettuato in quanto, come visto in altri esempi, quando il valore "best accuracy", cioè la massima accuratezza raggiunta dal modello durante l'addestramento, viene raggiunto lontano dal termine dell'addestramento, si potrebbe provare a settare il valore del parametro di addestramento 'epochs' (rappresenta la durata dell'addestramento) con l'esatto valore dell'epoca in cui si è raggiunta la best_accuracy. In molti casi il re-training della rete dopo questa modifica ha portato a prestazioni migliori senza overfittare.

Dopo aver effettuato questa analisi sono stati visualizzati i grafici di training/validation accuracy, roc curve, loss curve e la confusion matrix. In linea generale per ogni addestramento sia la curva di addestramento che quella di loss sono state caratterizzate da un andamento come in figura.



Step 4: Costruzione Keras Tuner

In questo step si è costruito un Tuner, cioè un ottimizzatore di iperparametri che risolve i punti deboli della ricerca di questi.

La costruzione è avvenuta tramite la funzione model_builder che effettua il tuning dei seguenti elementi:

- Numero di unità nel primo livello denso intermedio

- Decide se usare o meno un livello di Dropout che imposta casualmente le unità di input su 0 con una certa frequenza ad ogni passaggio durante il tempo di allenamento, il che aiuta a prevenire l'overfitting.

- Tuning del learning_rate tra [1e-2, 1e-3, 1e-4, 1e-5, 0.005, 0.05]

Successivamente è stato istanziato il sintonizzatore per eseguire l'hypertuning. Il Keras Tuner ha quattro sintonizzatori disponibili: RandomSearch , Hyperband , BayesianOptimization e Sklearn. Dopo aver effettuato vari tentativi l'Hyperband si è rivelato migliore degli altri.

Si crea quindi una chiamata 'EarlyStopping' per interrompere l'allenamento in anticipo dopo aver raggiunto un determinato valore per la perdita di convalida, questo valore viene settato con il parametro 'patience' che in questo caso è stato settato a 5, quindi dopo 5 epoche senza alcun miglioramento l'allenamento verrà interrotto.

Step 5: Riesecuzione Training

In questo ultimo step è stato rieseguito il training della rete neurale con i parametri scelti dal Tuner e sono state rivalutate le prestazioni, che si sono rivelate migliori del primo addestramento.

Anche in questo caso gli addestramenti eseguiti sui dataset aumentati con le Dipendenze Funzionali Rilassate hanno riportato migliori prestazioni rispetto alla baseline.

AdaBoost Classifier

L'ultimo modello implementato è un tipo di Ensemble Learning (noto anche come "meta-learning") che è stato inizialmente creato per aumentare l'efficienza dei classificatori binari. AdaBoost utilizza un approccio iterativo per imparare dagli errori dei classificatori deboli e trasformarli in forti.

I modelli **Ensemble Learning** utilizzano più algoritmi di apprendimento per ottenere prestazioni predittive migliori di quelle che potrebbero essere ottenute da uno qualsiasi degli algoritmi di apprendimento costitutivi da soli.

Questo tipo di classificatore è stato quello che rispetto agli altri due visti in precedenza ha ottenuto un accuracy del 100% sui dataset aumentati.

Step 1: Configurazione e Tuning dei parametri

In questo caso come primo step è stato effettuato il tuning degli iperparametri del modello attraverso GridSearch.

Step 2: Training e valutazione performance

Una volta ottenuti i best_params_ mediante GridSearch è stato eseguito il training del modello e ne sono state valutate le prestazioni.

5.4. Risultati

Di seguito si riporta la tabella dei risultati ottenuti con questi tre modelli di classificazione binaria addestrati su tutti i dataset. Da come si può notare solo AdaBoost, considerando le decisioni prese ed in settaggi applicati, ha consentito di ottenere un accuracy del 100% su tutti di dataset aumentati.

DATASET	kNN	MLP	ADABOOST
Dataset (baseline)	0.9279	0.9826 (best_acc) 0.9734 (evaluate)	0.9939
rfds_0	0.9318	0.9937 (best_acc) 0.9850 (evaluate)	1.0
rfds_1	0.9290	0.9882 (best_acc) 0.9745 (evaluate)	1.0
rfds_12	0.931818	0.9910 (best_acc) 0.9861 (evaluate)	1.0
rfds_2	0.9284	0.9882 (best_acc) 0.9867 (evaluate)	1.0
rfds_3	0.92849	0.9868 (best_acc) 0.9845 (evaluate)	1.0
rfds_4	0.92849223	0.9854 (best_acc) 0.9817 (evaluate)	1.0
rfds_8	0.92960	0.9861 (best_acc) 0.9834 (evaluate)	1.0

Analizziamo i modelli singolarmente:

Il modello kNN come vediamo è l'algoritmo che ha portato a performance più basse rispetto agli altri due modelli. Il motivo è probabilmente dovuto al fatto che il modello kNN in generale è un "lazy learner", ovvero è un modello che non fa molto durante il processo di training oltre a memorizzare semplicemente i dati. Si definisce un modello pigro perché gli altri modelli costruiscono un classificatore durante il training, come ad esempio AdaBoost che costruisce un ensemble classification model durante la fase di training; il modello kNN invece non costruisce alcun classificatore e nel momento in cui arriva un dato da classificare esamina i k elementi più vicini ed ottiene la classe cui dovrebbe appartenere il nuovo dato. Inoltre, è da precisare che kNN è il modello che ha impiegato più tempo per addestrarsi.

Quindi, tale modello esegue confronti di similarità per selezionale i "k vicini più vicini", quindi siccome l'aggiunta di feature aumenta la dimensionalità delle istanze, anche istanze più "vicine" potrebbero risultare "distanti". L'aggiunta di RFD permette quindi una migliore

discriminazione tuttavia, come si nota, il modello non raggiunge le performance degli altri due modelli.

La strategia di aggiunta delle RFD ha funzionato anche nel caso delle MLP, dove si è scelto di costruire, dopo vari tentativi, una rete composta da due livelli intermedi ed un numero di percettroni pari alla dimensione dell'input. I risultati analizzati attraverso le curve di perdita in fase di training e validation mostrano ottimi andamenti e non si sono verificati situazioni di overfitting. Tuttavia nonostate i buoni risultati, questo tipo di NN non è suggerita per un task di classificazione binaria per vari motivi: potrebbe risultate di difficile interpretazione, richiede troppi dati, potrebbe richiedere molto tempo per la creazione dell'architettura e per l'addestramento.

Infine, anche il modello AdaBoost migliora le sue performance a seguito dell'aggiunta delle RFD. AdaBoost è un modello di ensemble learning che combina più algoritmi di classificazione (come, ad esempio, gli alberi di decisione). L'obiettivo di questo modello è utilizzare un insieme di modelli deboli e combinarli per ottenere un modello molto performante. Le sue performance, superiori agli altri modelli, potrebbero essere date dal fatto che Adaboost è un modello meno inclide all'overfitting, facile da usare, con meno necessità di modificare i parametri a differenza di altri algoritmi come SVM ed estremamente flessibile ai vai tipi di dataset.

6. Secondo step: applicazione Feature Selection

Nel secondo step sono state applicate le tre tecniche di Feature Selection identificate.

Le tre tecniche selezionale sono:

1) Variance Filtering Method
2) Correlation Based Filtering Method
3) Boruta Method

Ognuna di queste tecniche è stata applicata nel seguente modo:

1) Sono state separate le 15 feature originali dalle RFD generale;
2) Si è applicata la metodologia di Feature Selection sulle RFD sia sul training set che sul test set;
3) L'insieme delle feature ottenuto è stato unito alle 15 feature originali inizialmente separate;
4) Si è rieseguito il tuning degli iperparametri dei rispettivi 3 modelli, implementati nel primo step del progetto, sul nuovo training set ottenuto;
5) Sono state valutate le performance dei singoli modelli sul nuovo test set, comparando i risultati (accuracy, precision, recall, f1) con il modello iniziale addestrato con tutte le RFD.

6.1. Variance Filtering Method

Per eseguire questa tecnica in primis è stato diviso l'intero dataset in dataset di training e dataset di testing secondo il criterio 80/20.

In questa tecnica sono state eliminate le colonne che rappresentano feature:

- Constant Feature: feature i cui valori contenuti all'interno delle colonne sono tutti uguali; quindi, hanno una varianza pari a 0.

- Quasi-Constant Feature: feature i cui valori contenuti all'interno delle colonne sono quali tutti uguali, per quasi si intende che hanno una varianza del 10 % massimo.
- Duplicate Feature: feature duplicate, ovvero coppie di colonne che hanno stessi valori nelle celle.

Una volta identificati questi tipi di colonne è stato eseguito il filtraggio eliminandone sia dal dataset di training che dal dataset di testing. Infine, le feature rimanenti sono state di nuovo unite alle 15 feature di baseline e sono stati riaddestrati i modelli.

6.2. Correlation Based Filtering Method

Questa tecnica consiste nel rimuovere quelle features che sono altamente correlate. Per identificare le feature altamente correlate è stata utilizzata la matrice di correlazione, la quale contiene in ogni cella l'indice di correlazione Pearson che può essere compreso tra -1 e 1. Un valore vicino all'1 indica che esiste una correlazione positiva tra le due feature, mentre un valore vicino al -1 indica un correlazione negativa tra le due feature. Un valore pari a 0 indica l'assenza di correlazione tra le feature.

La difficoltà in questa tecnica è stabili oltra quale soglia bisogna classificare delle feature come "correlate" e di conseguenza eliminarle. Per risolte questo problema sono state eseguite 5 selezioni diverse con valori di soglia pari a 0.5, 0.6, 0.7, 0.8, 0.9. Quindi per ogni valore di soglia t è stata letta l'intera matrice di correlazione e sono state eliminate quelle feature che hanno un indice di correlazione maggiore di t o minore di -t.

Successivamente le feature ottenute nei cinque casi sono state unite alle 15 feature di base sia nel dataset di training che nel dataset di testing. Infine, stati eseguiti cinque addestramenti per ogni modello implementato ed è stato, quindi, selezionato l'insieme di feature con la soglia t_i che ha portato a migliori performance per quel singolo modello.

6.3. Boruta Method

Questo metodo si basa sulla costruzione di "shadow feature". In pratica, partendo dal dataset di training, viene creato un altro dataset mescolando casualmente ogni caratteristica. Queste funzioni permutate sono chiamate **funzioni ombra** . A questo punto, il dataset shadow viene collegato al datset originale per ottenere un nuovo dataset (lo chiameremo *X_boruta*), che ha il doppio del numero di colonne del dataset originale. Successivamente viene addestrato un classificatore Random Forest sul dataset ottenuto e vengono calcolati i valori di importanza per ogni colonna, questo punteggio viene detto "Z-Score".

Quindi successivamente, si confronta l'importanza di ciascuna feature originale con la migliore feature ombra, ovvero la feature ombra che ha uno Z-Score più alto (Z-Score max); se il valore d'importanza della feature originale è superiore allo Z-Score max allora questa verrà selezionata come importante per il task di classificazione.

Dopo aver ottenuto la lista delle feature selezionate, è stato visualizzato la lista dei ranking: un valore pari a 1 indica rank massimo, cioè che la feature è stata selezionata invece, per valori >1, più questo è alto più l'importanza della feature è bassa.

Come ultimo passo sono stati rieseguiti tutti gli addestramenti con le sole feature identificate con rank pari ad 1.

6.4. Risultati

Per valutare gli effetti di queste tecniche di Feature Selection sono stati eseguiti gli addestramenti dei tre modelli kNN, MLP e AdaBoost su due dataset: rfds_0 e rfds_12.

Di seguito possiamo analizzare i risultati ottenuti dalla sperimentazione effettuata. Si riporta una prima tabella contenente metriche accuracy, precision e recall per i tre modelli addestrati su tutte le RFD; successivamente si riporta una tabella per modello contenente le medesime metriche per ogni tecnica di Feature Selection applicata. I valori di Precision e Recall fanno riferimento ai valori della classe 'FAKE' (riga 0) del report di classificazione.

Nell'ultima sezione dei risultati è possibile trovare le confusion matrix raccolte per tutti i singoli addestramenti eseguiti nella fase di sperimentazione.

DATASET	kNN			MLP			ADABOOST		
	P(s)	R(s)	A(s)	P(s)	R(s)	A(s)	P(s)	R(s)	A(s)
rfds_0 (377 feat)	0.85	0.97	0.9318	0.98	0.99	0.9850 (evaluate) 0.9937 (best_acc)	1.0	1.0	1.0
rfds_12 (226 feat)	0.85	0.97	0.9319	0.98	0.98	0.9861 (evaluate) 0.9910 (best_acc)	1.0	1.0	1.0

Modelli addestrati su tutte le RFD

Nelle cella A(s) delle tabelle sottostanti riportiamo il valore di Accuracy con il relativo numero di feature selezionate dalla rispettiva tecnica di Feature Selection. Nel caso della Correlation Filtering, si riporta anche la soglia migliore di Correlazione che è stata selezionata per eseguire il filtraggio delle feature.

kNN									
DATASET	Variance Filtering			Correlation Filtering			Boruta		
	P(s)	R(s)	A(s)	P(s)	R(s)	A(s)	P(s)	R(s)	A(s)
rfds_0 (377 feat)	0.86	0.97	(53 feat) 0.93	0.86	0.97	(best with 115, 0.9t) 0.93	0.99	0.98	(127 feat) 0.9878
rfds_12 (226 feat)	0.86	0.97	(73 feat) 0.93	0.86	0.97	(best with 174, 0.9 t.) 0.93	0.99	0.98	(118 feat) 0.99

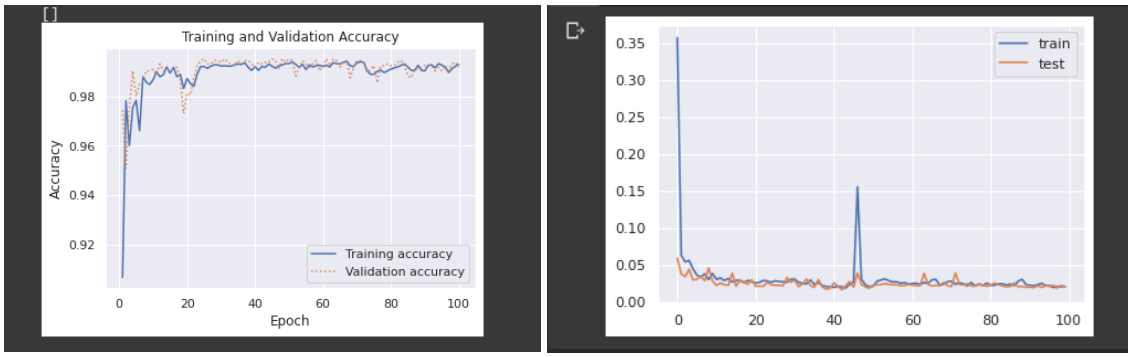
MLP									
DATASET	Variance Filtering			Correlation Filtering			Boruta		
	P(s)	R(s)	A(s)	P(s)	R(s)	A(s)	P(s)	R(s)	A(s)
rfds_0	0.97	0.98	(53 feat) 0.9872	0.98	0.98	(best with 101, 0.8t) 0.9878	0.99	0.98	(127 feat) 0.9889
rfds_12	0.98	0.98	(73 feat) 0.98	0.99	0.98	(best with 171, 0.9t) 0.9872	0.99	0.98	(118 feat) Best acc 1.0 Eval 0.99

AdaBoost									
DATASET	Variance Filtering			Correlation Filtering			Boruta		
	P(s)	R(s)	A(s)	P(s)	R(s)	A(s)	P(s)	R(s)	A(s)
rfds_0	1.0	1.0	(73 feat) 1.0	1.0	1.0	(best with 70, 0.5t) 1.0	1.0	1.0	(127 feat) 1.0
rfds_12	1.0	1.0	(73 feat) 1.0	1.0	1.0	(best with 118, 0.5t) 1.0	1.0	1.0	(118 feat) 1.0

Analizziamo le prestazioni sui singoli modelli:

La prima tecnica, Variance Filtering, ha portato ad una selezione di 53 feature su 377 (rfds_0) e 73 feature su 226 (rfds_12). Dalle metriche raccolte nelle tabella notiamo che, da un punto di vista dall'Accuracy (A(s)), i valori sono equiparabili ai risultati ottenuti addestrando i modelli su tutte le RFD; mentre, nel caso del modelli kNN abbiamo un miglioramento della Precision su entrambi i dataset ed invece un peggioramento delle stessa sul dataset rfds_0 addestrato con MLP. Questi cambiamenti sono dovuti alla riduzione dei Falsi Positivi e Falsi Negativi nel caso di kNN ed all'aumenti di questi nel caso di MLP (solo sul dataset rfds_0).

Possiamo notare che il metodo Variance Filtering è quello che effettua una selezione più restrittiva delle feature arrivando ad eliminare circa l'86% delle feature nel dataset rfds_0. Questo ci fa capire che tra le tante RFD generale molte sono feature constanti o duplicate che singolarmente non avrebbero contribuito al miglioramento del modello. È da precisare il grande svantaggio di questo metodo, ovvero il fatto che si tratti di un metodo statico, cioè vengono considerate le feature singolarmente; quindi, se due feature singolarmente non sono significative ma combinate si, con questa tecnica si perderebbero comunque. La tecnica V.F. non ha portato a situazioni di overfitting in nessun caso e per il modello MLP non si sono riscontrate curve di loss in Training/Validation alterate rispetto all'addestramento originale (in figura è possibile vedere un andamento comune per l'addestramento della rete a seguito della selezione delle feature).



Con il secondo metodo, Correlation Filtering, notiamo un numero di feature più alto rispetto al Variance Filtering, questo perché il metodo Correlation Filtering considera le relazioni tra tutte le feature indipendenti. Anche in questo caso possiamo notare un miglioramento della Precision con il modello kNN, mentre con il modello MLP si ha un abbassamento della Recall sul dataset rfs_0 ed un miglioramento della Precision sul dataset rfds_12 (possiamo vedere nella confusion matrix un azzeramento dei Falsi Positivi ma un aumento dei Falsi Negativi). Il metodo Correlation FIltering eseguito sullo stesso dataset ha portato a sottoinsiemi di feature diversi a seconda del modello. Ad esempio, sul dataset rfds_12 con il modello kNN la soglia di correlazione migliore è stata 0.9 con 174 feature mentre con AdaBoost la soglia migliore è stata 0.5 con 118 feature selezionate.

Il metodo Boruta, invece, è stato il modello allo stesso tempo più costoso da un punto di vista computazionale ma anche più performante, in quanto ha migliorato i valori di Precision, Recall ed Accuracy sia sul modello kNN che su MLP (che è riuscito a raggiungere anche il 100% come best accurary durante l'addestramento), con una notevole riduzione delle feature, che è stata in media del 57% sul totale delle RFD.

Il modello AdaBoost non ha riportato peggioramenti delle performance a seguito delle tecniche di Feature Selection applicate.

Tuttavia, per tutti i modelli c'è stato una notevole riduzione del tempo di addestramento e della complessità del modello stesso, dovuto alla marcata selezione di feature effettuata.

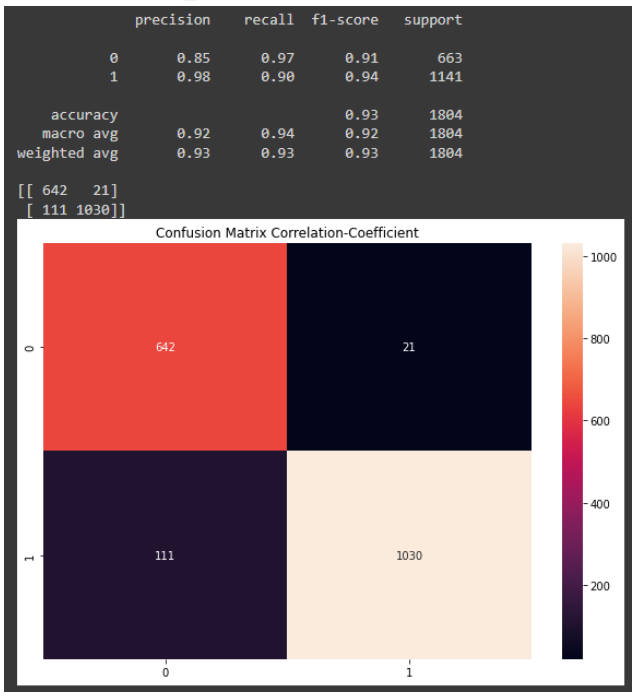
Si riporta una ulteriore osservazione fatta durante i vari addestramenti.

Addestrando il modello kNN sui dataset senza le 15 feature principali, ma solo con le RFD selezione dalle tecniche Variance Filtering e Correlation Filtering, il modello ha ottenuto migliori performance rispetto agli addestramenti eseguiti con le RFD selezionate unite alle 15 feature originali. In particolare, kNN ha raggiunto il 100% di accuracy.

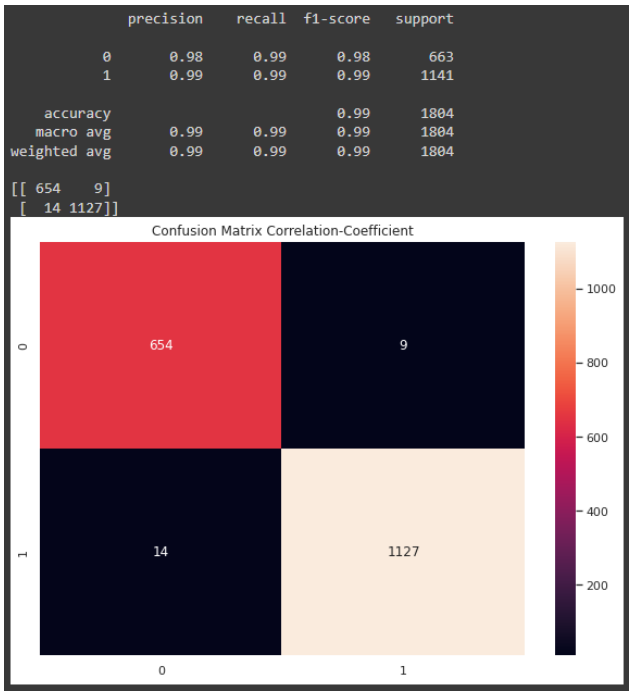
Questo comportamento ci fa capire che, anche in presenza di feature che rappresentano rumore in un dataset, è possibile inferire RFD le cui feature associate, introducono un'ulteriore informazione semantica per ogni account, il quale facilita il task di classificazione dei modelli. Inoltre, in termini di qualità delle feature, utilizzando solo le feature aggiuntive (RFD) si riescono a migliorare le performance, risolvendo eventualmente problemi di overfitting.

6.4.1. Confusion Matrix (iniziali)

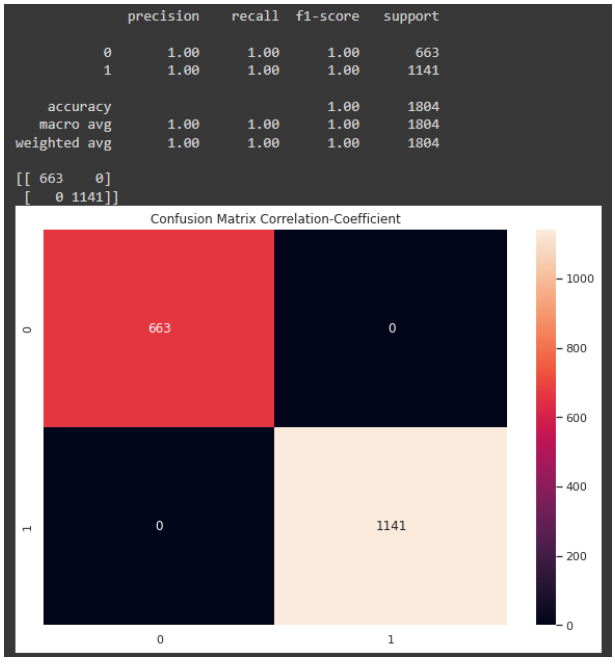
rfds_0



kNN

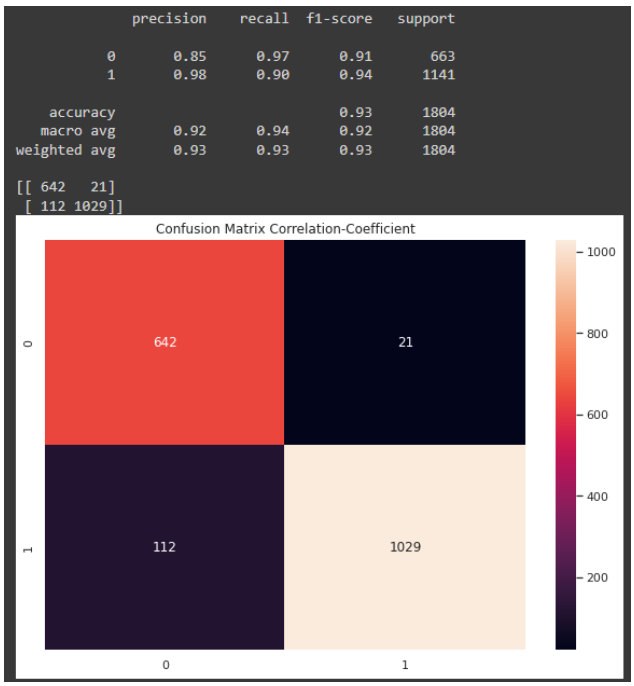


MLP

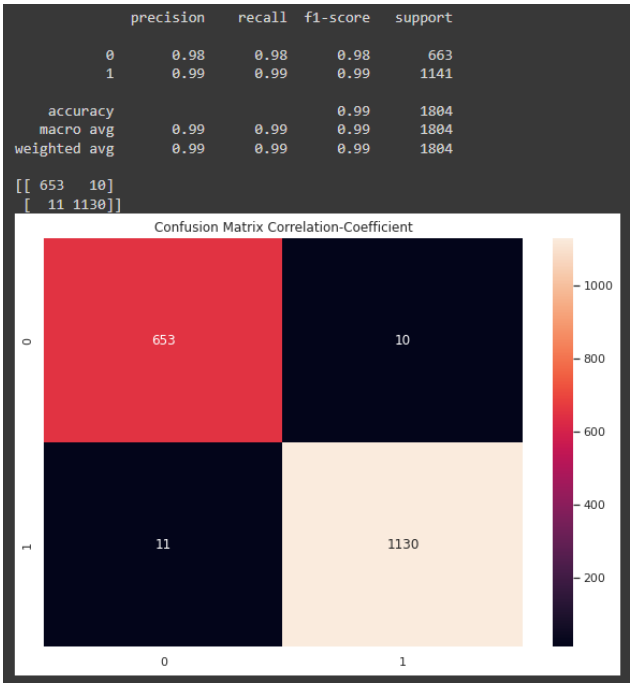


AdaBoost

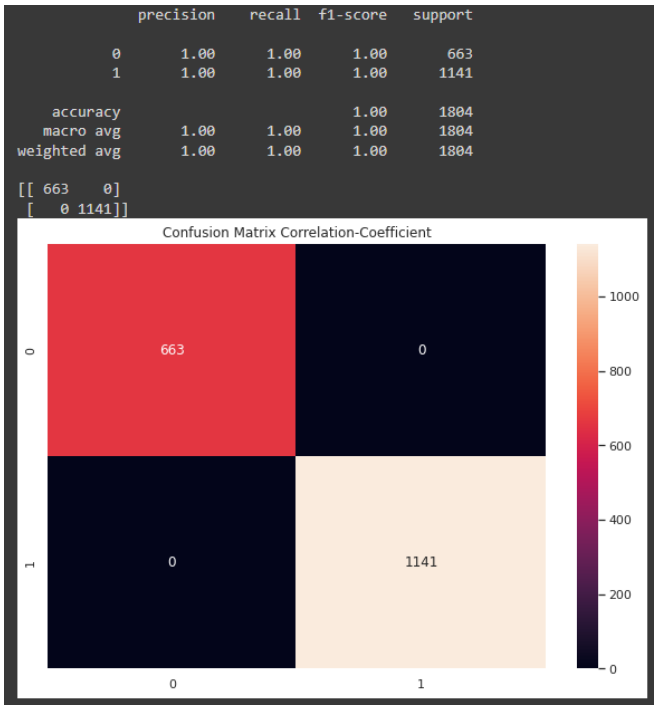
rfds_12



kNN



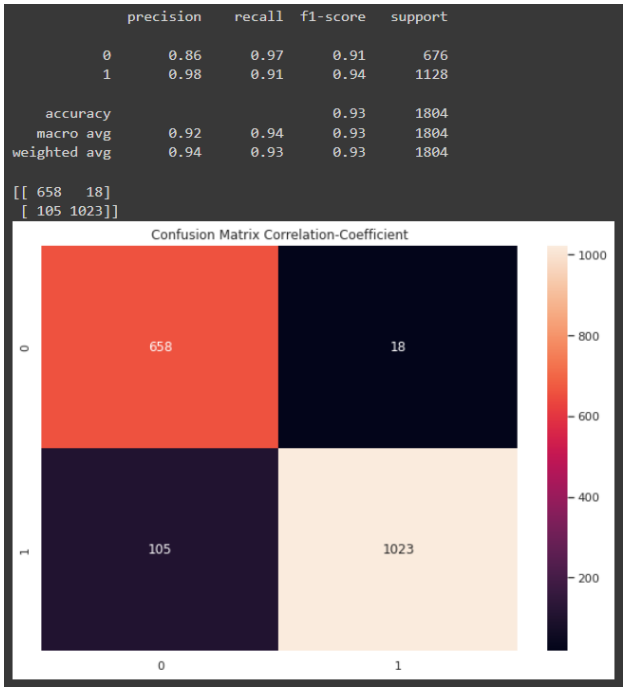
MLP



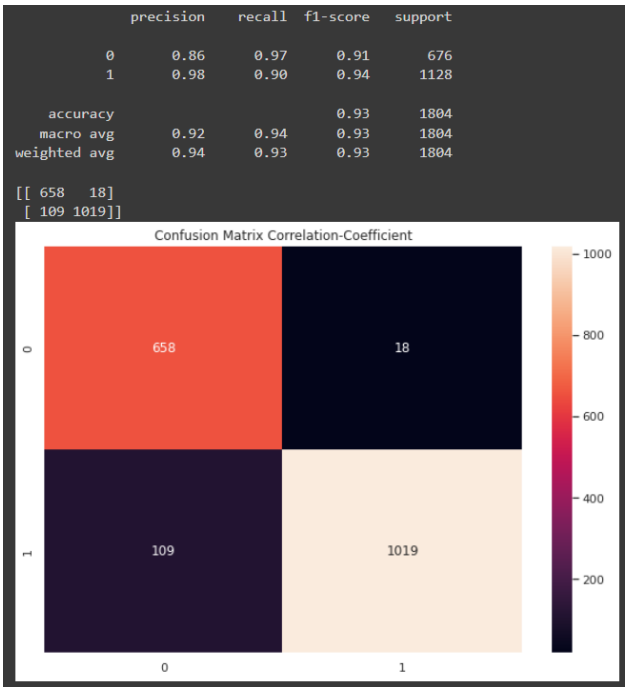
AdaBoost

6.4.2. Confusion Matrix (post Feature Selection)

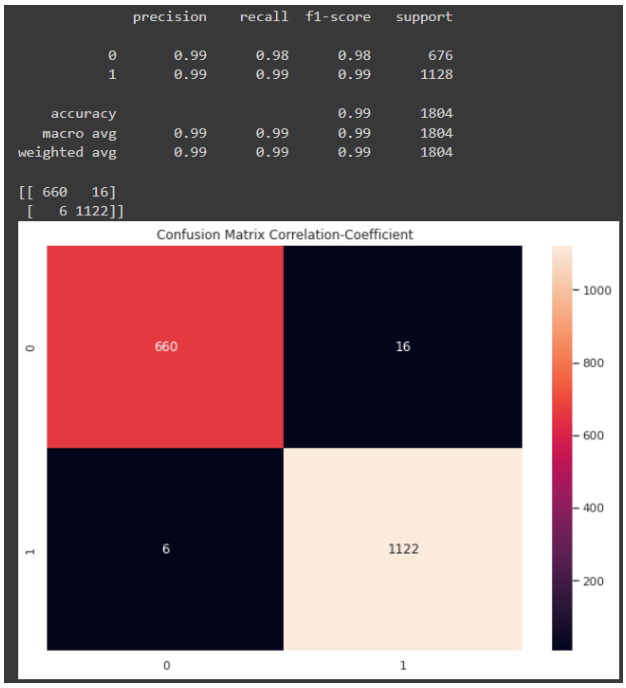
rfds_0_kNN



Variance Filtering

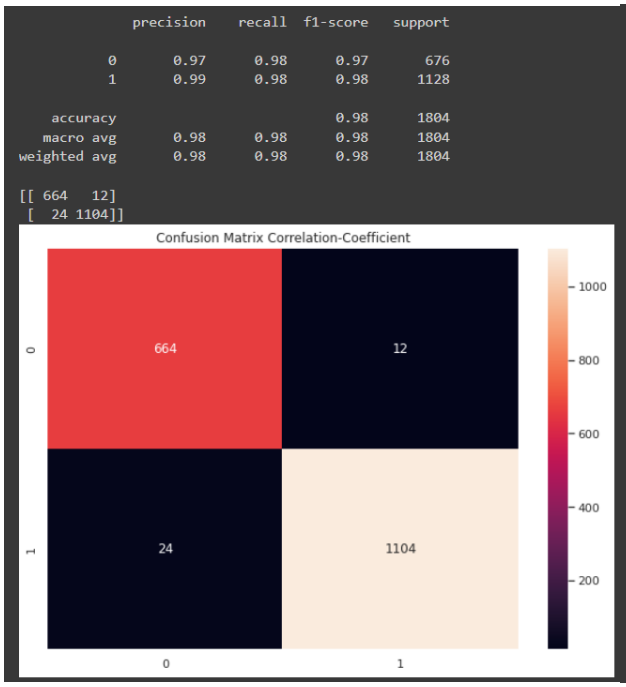


Correlation Filtering

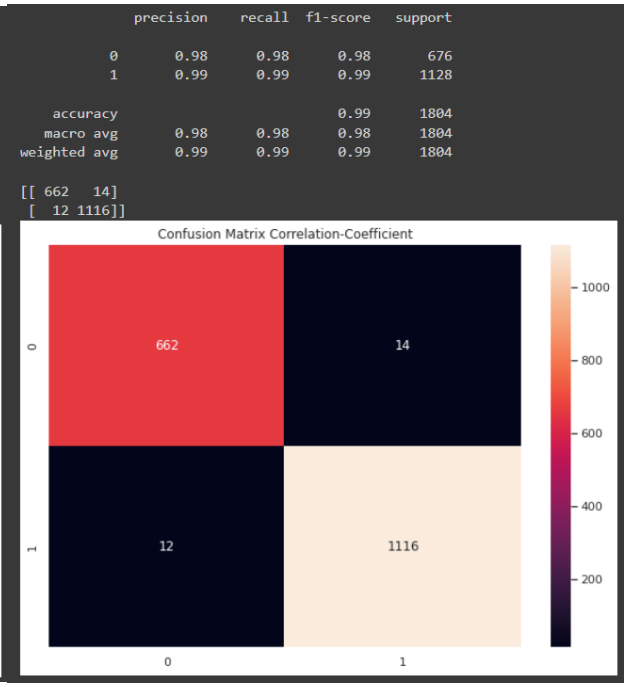


Boruta

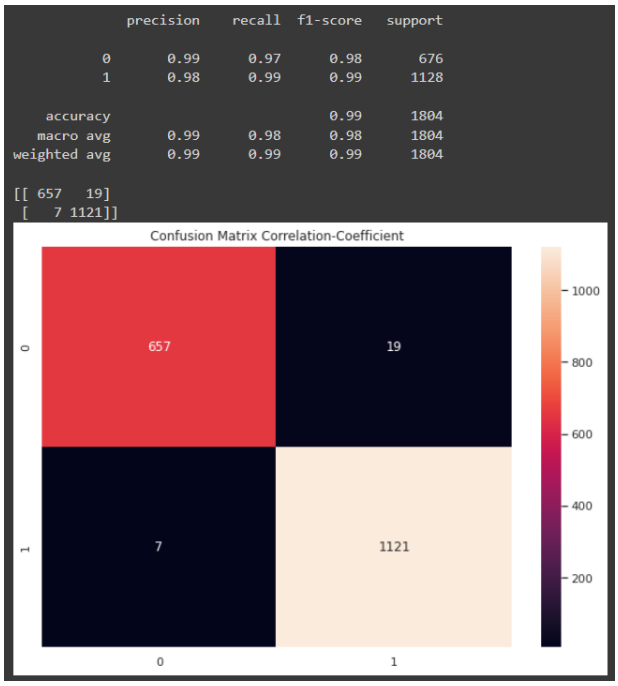
rfds_0_MLP



Variance Filtering

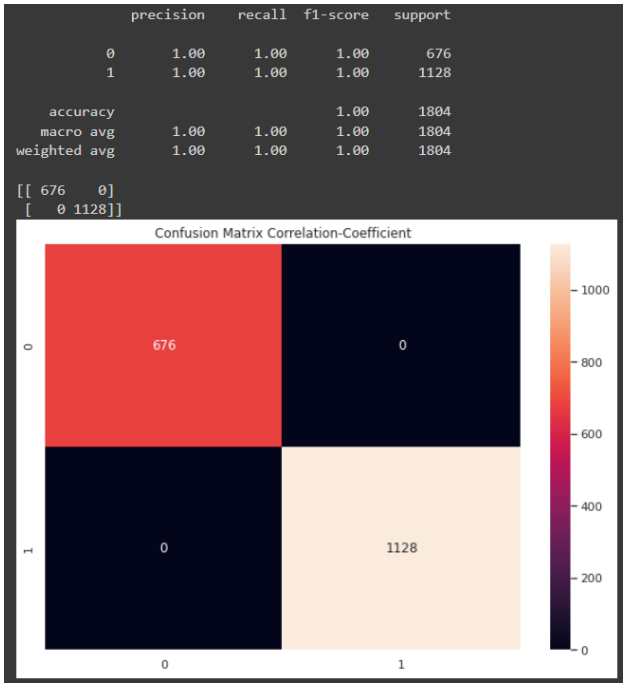


Correlation Filtering

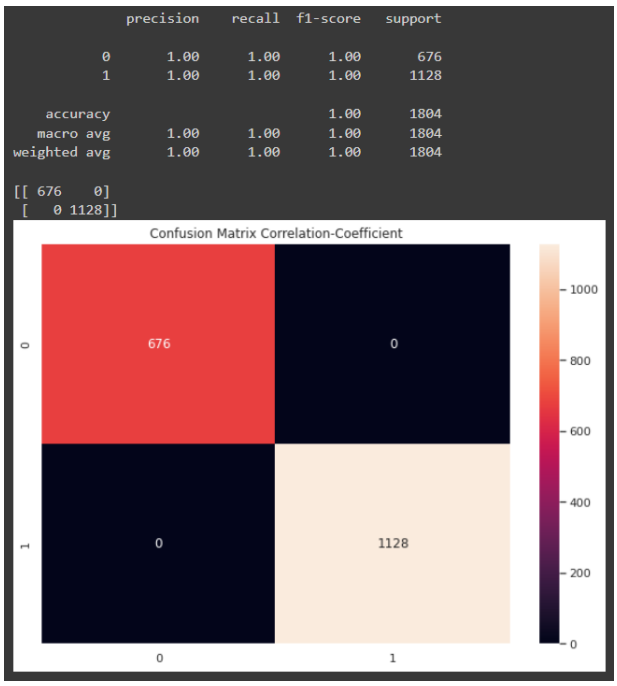


Boruta

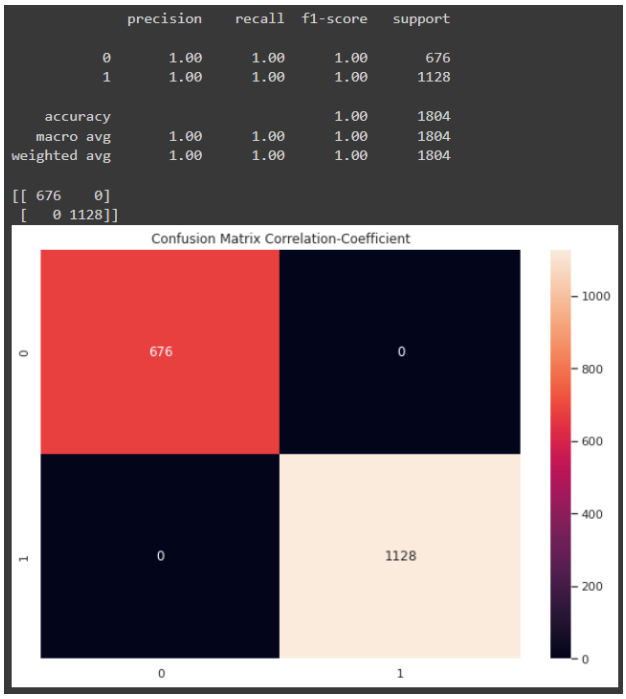
rfds_0_AdaBoost



Variance Filtering

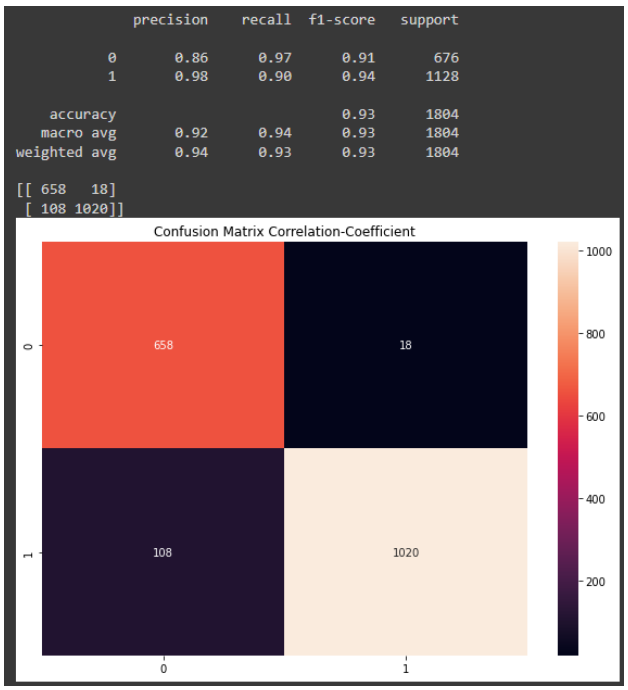


Correlation Filtering

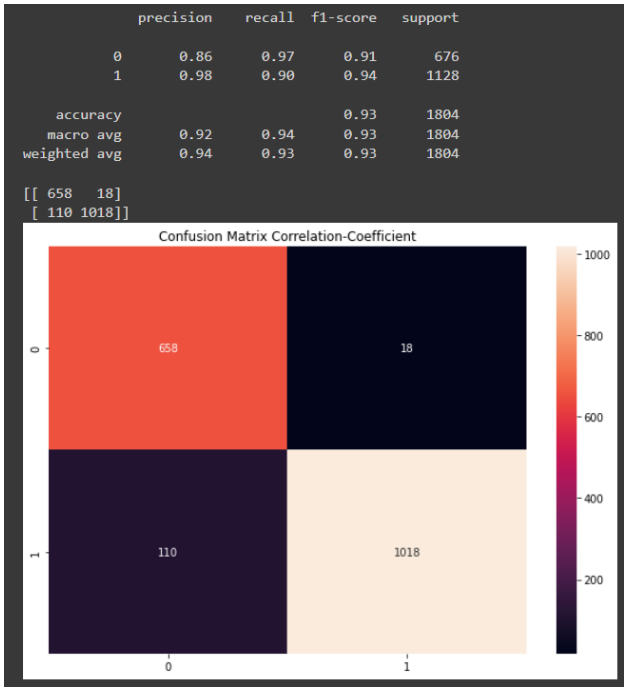


Boruta

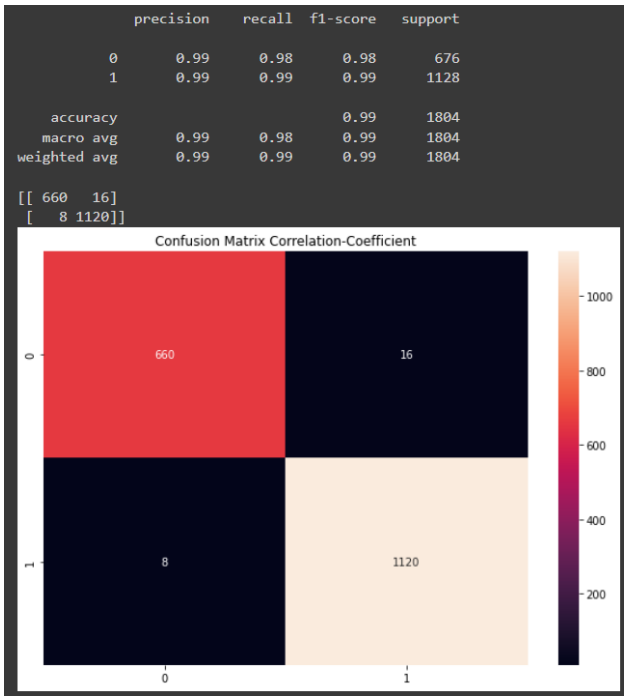
rfds_12_kNN



Variance Filtering

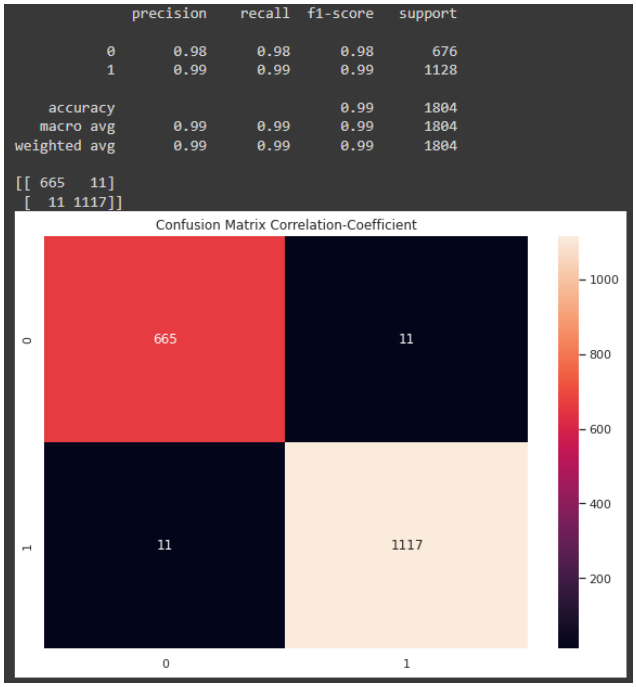


Correlation Filtering

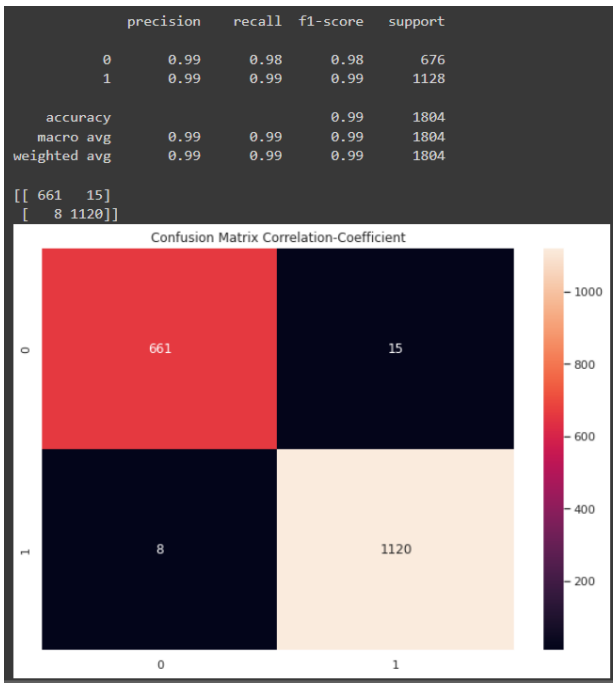


Boruta

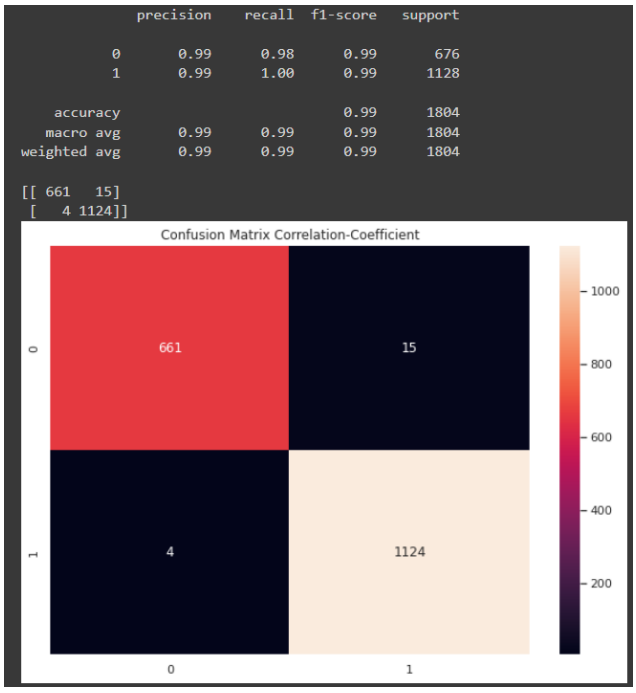
rfds_12_MLP



Variance Filtering

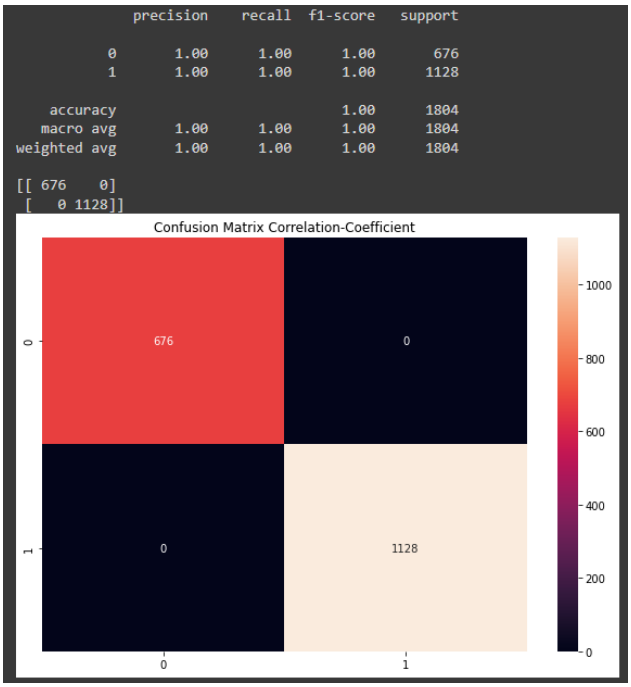


Correlation Filtering

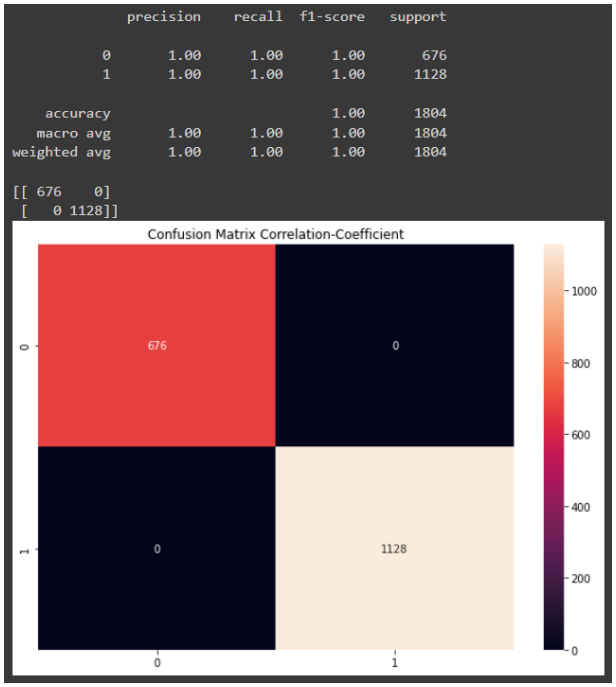


Boruta

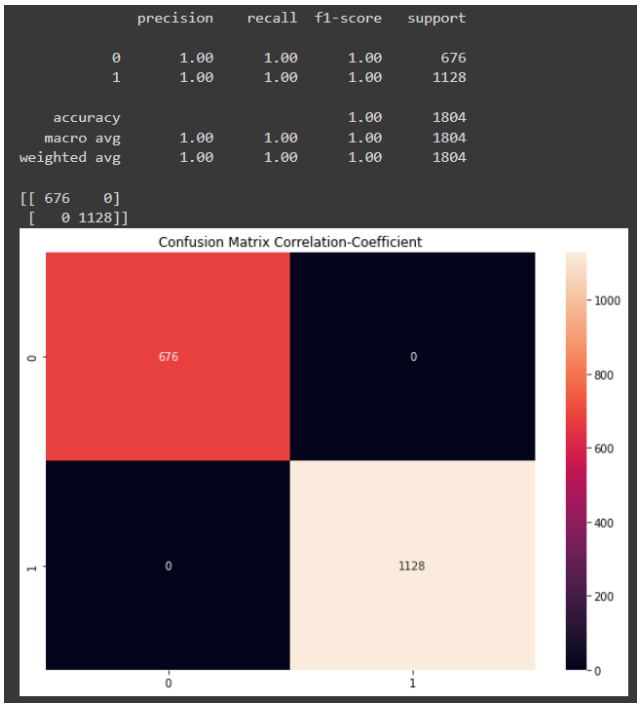
rfds_12_AdaBoost



Variance Filtering



Correlation Filtering



Boruta

7. Conclusioni

Effettuare operazioni di Feature Selection è uno step di data prepocessing molto importante ed è bene conoscere quali tecniche siano più adatte a seconda del task. Le tecniche sperimentate hanno portato, nel peggiore dei casi, a performance equiparabili, permettendo quindi di ridurre notevolmente il numero di feature e di conseguenza la complessità dei modelli.

Tuttavia, risulta necessario effettuare altri test su dataset diversi contenenti RFD per avere una comparazione più ampia. Bisogna precisare che, anche senza aggiungere RFD al dataset di baseline, tutti i modelli di classificazione hanno ottenuto ottimi risultati; potrebbe risultare interessante ai fini della ricerca testare queste tecniche su dataset che ottengono delle performance minori.