

Progetto MOBD

Corso di Laurea Magistrale in Ingegneria Informatica

Data Science & Engineering

Anno accademico 2021-2022

Giulio Appetito 0321669

Anastasia Brinati 0321654

Indice

1. Scopo di questo elaborato
2. Analisi
   1. Training e Test
   2. Preprocessing
      1. Encoding
      2. Feature Importance & Feature Selection
      3. Sostituzione Valori Mancanti
      4. Scaling
      5. Anomaly Detection
      6. Balancing
3. Considerazioni conclusive
4. **Scopo di questo elaborato**

La seguente relazione ha l’obiettivo di illustrare il progetto per l’esame di Metodi di Ottimizzazione per Big Data, entrando nel dettaglio dei passi seguiti per l’addestramento di un classificatore al fine di risolvere un problema di classificazione binaria ed ottenere una buona soglia di accuratezza.

Abbiamo utilizzato Google Colab per la stesura del codice, utilizzando librerie di Data Analysis e Machine Learning quali Numpy, Pandas ed Sklearn, già presenti nell’ambiente di lavoro.

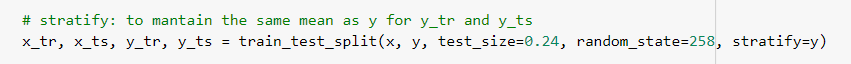
1. **Analisi**

L’analisi ha seguito la pipeline presentata a lezione basata sul CRISP-DM, a partire dalla fase di Data Preparation.

* 1. **Training e Test Set**

In primo luogo, abbiamo suddiviso il dataset ‘train.csv’ in due sottoinsiemi, uno di training ed uno di test, per poter procedere all’addestramento di un classificatore tramite il primo insieme, e poi testarne le performance con il secondo set ottenuto, usando quest’ultimo come validation. La suddivisione è avvenuta utilizzando la funzione train\_test\_split(x, y, test\_size = 0.3), dalla quale abbiamo ottenuto, inizialmente con un rapporto 70-30, sia il set di training con le relative etichette in un vettore separato, sia il test set allo stesso modo. Inoltre abbiamo utilizzato il parametro stratify=y per ottenere y di test e di training distribuite con la stessa media.

Diversi risultati empirici ci hanno portato a considerare come divisione più conveniente test\_size = 0.24.



* 1. **Preprocessing**

A seguito della separazione ci siamo occupati di misurare quale fosse il grado di sbilanciamento tra le due classi, che è risultato essere circa 0,7591 a 0,2408.

Il training set fornitoci consiste di 14 features, indicate come F0 - F13, sia di tipo continuo che categorico, per cui la prima questione che abbiamo affrontato è stata la conversione dei valori di queste ultime.

* + 1. **Encoding**

Dapprima abbiamo ragionato su come convertire in valori numerici le features categoriche, che corrispondevano a ben 8 attributi su 14 totali. Per questo scopo è stata utilizzata la tecnica one hot encoding, ed abbiamo applicato questa conversione da categorical data a features numeriche sulle colonne F1, F3, F5, F6, F7, F8, F9 ed F13, utilizzando la funzione get\_dummies().

Immagine che contiene testo

Descrizione generata automaticamente

Non potendo risalire al significato delle features, tranne alcune come il genere, la nazionalità, la ‘razza’ e – crediamo - l’età, abbiamo fatto alcune assunzioni per valutare se impattassero le performance, una volta completato il training:

* per quanto riguarda il genere, abbiamo optato per lasciare la conversione in due colonne ‘Female’ e ‘Male’, binarie e mutuamente esclusive, piuttosto che una sola feature binaria;
* la colonna F3 (categorica) abbiamo notato essere strettamente correlata alla feature F4 (continua), infatti i valori Ki della prima corrispondevano 1 ad 1 con i valori numerici della seconda, per cui il tentativo è stato cancellare completamente la feature F3, che però non è risultato migliore di applicare anche su di essa il processo di encoding, trasformandola dunque in altre 6 colonne binarie (e anch’esse mutuamente esclusive), o ancora abbiamo provato ad eseguire l’encoding su F3 e a cancellare F4, ma anche questo non ci ha fatto raggiungere risultati migliori; infine abbiamo optato per un codificare ogni valore di F3 con il corrispettivo in F4: ;
* ragionamento simile al precedente è stato fatto anche sugli attributi F10 ed F11 che risultano assumere entrambe il valore ‘0’ rispettivamente per il 91% e 95% delle istanze, tuttavia questo è stato il caso in cui la presenza delle due colonne è risultata fondamentale e si è notato un calo evidente nell’accuratezza;
* altro cambiamento riguarda la colonna F8, dove abbiamo notato un aumento di accuratezza nel sostituire le ‘razze’ diverse da ‘caucasian’ con ‘non-caucasian’.

Diverse considerazioni sono avvenute sulla feature F13, in quanto, anche in questo caso, una grandissima parte del set iniziale presenta il valore ‘USA’, per cui si è provato a verificare se eliminando questa colonna si ottenessero risultati migliori, ma non è stato così.

Inizialmente abbiamo effettuato l’encoding sul dataset di partenza senza modificare le istanze, ma questo ci ha portato a lavorare su un dataframe di 104 colonne totali.

Allora abbiamo cercato di applicare varie trasformazioni di sostituzione delle nazioni:

* con il continente di appartenenza, ottenendo 4 colonne (AmericaN, AmericaS, Europa e Asia) e non più (circa) 38, il che ci ha permesso di velocizzare i tempi di esecuzione;
* (USA, non-USA);
* (America, Europa, Asia);
* Suddivisione 1°, 2° e 3° mondo;

Ed il miglior outcome è sortito dalla scelta USA e non-USA.

Ovviamente le scelte fatte per la trasformazione del set di colonne sono state ripetute sia sul test set usato per la valutazione eseguita da noi, che nella routine di test.

L’encoding è avvenuta tramite l’oggetto ‘MyEncoder’ che abbiamo definito nella routine di training:

Immagine che contiene tavolo

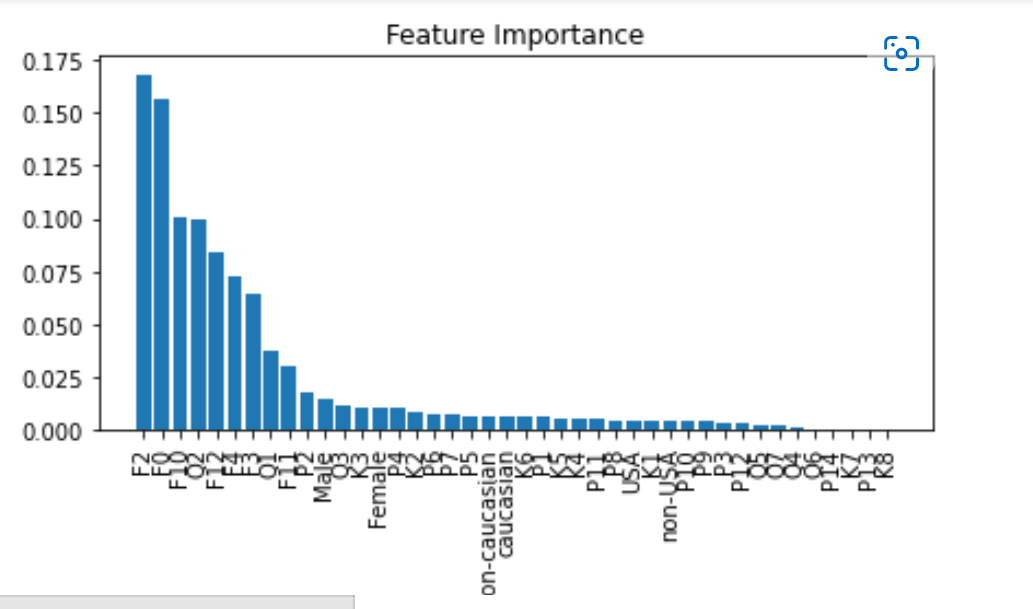
Descrizione generata automaticamente

Questo encoder da noi realizzato prevede le funzioni ‘fit’ e ‘transform’, e dunque è stato inserito nella pipeline di preprocessing che viene utilizzata nella routine di testing, di modo da lasciare al tester solamente il compito di eseguire la transform sul test set, senza effettuare modifiche ed eseguire conversioni su quest’ultimo, né tantomeno ricorrere al training set in maniera esplicita.

* + 1. **Feature Importance e Feature Selection**

Abbiamo tentato di utilizzare la feature selection per evidenziare le feature più e meno significative, ai fini di un addestramento con alta accuracy, tuttavia gli unici cambiamenti significativi hanno riguardato l’esclusione della feature F7.

Al fine di evidenziare le feature più significative abbiamo tracciato il seguente grafico, grazie alla libreria matplotlib:



* + 1. **Sostituzione Valori Mancanti**

A seguito di questi ragionamenti, abbiamo guardato ai valori mancati nel dataset **iniziale**, rispettivamente:

F1: 1836, F6: 1843, F13: 583, tutti e tre relativi a features categoriche. Dato che il metodo get\_dummies() considera i valori NaN come elementi nel dominio di un determinato attributo, questi vengono rappresentati da una ulteriore colonna. Le opzioni che abbiamo considerato a questo punto sono state:

* scartare le tuple con l’insieme di valori mancanti, prima dell’encoding;
* andare ad imputare dei valori validi sostituibili, magari con algoritmi che non guardano alla media di ogni singola feature ma alle istanze nel complesso, tipo KNN, tuttavia andando ad introdurre parecchio rumore, dato che ad esempio nel caso della colonna F6, ci saremmo ritrovati da una tupla con 1 NaN ad una con 16 NaN (P1, ..., P16) da stimare (senza considerare che non sarebbe neanche stato possibile assicurare la mutua esclusitività);
* eliminare le colonne NaN generate dall’encoding, lasciando tuple con tutti ‘zeri’ in corrispondenza di ogni possibile valore assumibile dalla feature, senza alcuna motivazione;

La strategia da noi adottata è stata utilizzare un SimpleImputer che andasse a sostituire i valori NaN con il ‘most\_frequent’, non appena suddiviso il dataset in test e training, **prima** di effettuare l’encoding. Per quanto riguarda il test set i valori mancanti sono stati sostituiti con il most frequent calcolato sul training set.

Nella routine di test abbiamo utilizzato una pipeline il cui primo step è proprio l’istanza di imputer con cui si è effettuata la sostituzione dei valori mancanti nella routine di training.

Immagine che contiene testo, interni, screenshot

Descrizione generata automaticamente

* + 1. **Scaling**

Lo step successivo di cui ci siamo occupati è stato lo scaling delle features, in quanto anche le stesse features continue erano molto differenti per unità di grandezza, ad esempio F0 (età) ed F2.

Lo scaler da noi utilizzato è RobustScaler(), il quale utilizza statistiche che sono robuste agli outliers, tuttavia abbiamo provato su vari training set (variando random seed) se altri transformers fossero più performanti, quali ad esempio MinMaxScaler e StandardScaler, ma così non è risultato.

Come ultimo step della pipeline per il preprocessing nella routine di test abbiamo inserito lo scaler adottato durante il training.

* + 1. **Anomaly Detection**

Utilizzando IsolationForest, andiamo a eliminare le righe del training set che risultano anomale, sebbene questo step sia risultato nell’esclusione di un numero irrisorio di valori. Questo comportamento è stato riscontrato su quasi la totalità dei training set che abbiamo considerato (cambiando il random seed).

Immagine che contiene testo

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamente

Poiché non sicuri di star seguendo la scelta giusta nell’applicare l’anomaly detection per eliminare così pochi valori, abbiamo sempre provato ad addestrare un classificatore su un training set sul quale non fosse applicato questo step.

* + 1. **Balancing**

Essendo il dataset di partenza molto sbilanciato, abbiamo testato varie tecniche per il balancing.

**Immagine che contiene testo

Descrizione generata automaticamente**

Da subito abbiamo optato per un undersampling, provando sia con RandomUnderSampler che con altri transformers che adottano algoritmi più complessi, quali ad esempio ClusterCentroids o NearMiss. Abbiamo applicato il sampling in modo che avvenisse successivamente all’anomaly detection, in quanto per questa ulteriore fase abbiamo deciso di utilizzare ‘IsolationForest’ che necessita di più dati possibile per l’addestramento. L’undersampling ha portato tuttavia a risultati ben al di sotto del benchmark dell’80% di accuratezza; dunque, abbiamo iniziato a ragionare sull’oversampling. Testando sia RandomOverSampler che algoritmi SMOTE, il sampler che ci ha garantito le performance migliori è stato KMeansSMOTE.

Per concludere i ragionamenti sul balancing, abbiamo considerato la possibilità di eseguire sia oversampling, in diverse percentuali, che undersampling (separati o meno dall’anomaly detection): il risultato è stato non migliore di quelli/o ottenuto effettuando solamente la fase di oversampling. Riportiamo di seguito diverse prove che abbiamo eseguito confrontando sulla stessa tipologia di classificatore, con gli stessi parametri, e sullo stesso training set, come variasse l’applicazione o meno degli step di anomaly detection e undersampling:

Immagine che contiene testo

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamenteImmagine che contiene testo, quotidiano, ricevuta, screenshot

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamenteImmagine che contiene testo, ricevuta, screenshot

Descrizione generata automaticamente

* + 1. **Scelta Classificatore**

Abbiamo testato l’addestramento di diversi classificatori:

* SVC: nonostante aver manipolato i vari parametri, in particolare concentrandoci sul variare la funzione kernel(rbf, sigmoid, poly) ed i valori C[0.01, 0.1, 1.0, 10.0] e gamma[0.01, 0.1, 0.3, 0.5, 1.0, auto], non ha fornito risultati sufficienti.
* RandomForestClassifier: che inizalmente ci ha fornito ottime performance, e lo abbiamo testato su profondità[2,6] e numero di estimators[40,1500].
* AdaBoostClassifier: dopo gradient boosting è risultato essere il classificatore con accuracy migliore.
* GradientBoostingClassifier: è risultato essere quello che ha fornito score migliori, sia per quanto riguarda accuracy, sia l’F1 score.

Per effettuare il tuning dei parametri dei vari classificatori abbiamo fatto uso della Grid Search cross-validation, fornita da Scikit-Learn, alternata alla RandomizedSearchCV; quest’ultima soluzione è risultata necessaria a casua del tempo computazionale eccessivamente elevato richiesto dalla GS-cv, per via dell’elevato numero di fit che l’algoritmo deve effettuare prima di completare.

Sono stati dunque effettuati diversi tentativi euristici, che ci hanno permesso di ottenere risultati a volte migliori rispetto a quelli forniti dalle precedenti ricerche (accettando un tradeoff tra accuracy ottenuta e tempo richiesto per la computazione), e ci hanno inoltre portato a selezionare il classificatore.

1. **Considerazioni Conclusive**

Dato che il progetto prevedeva un benchmark sull’accuratezza, riteniamo che le due classi del problema avessero la stessa ‘importanza’, per cui una stima basata su questa metrica è ovviamente sensata.

D’altra parte, non avendo questa certezza assoluta, abbiamo confrontato varie metriche, quali l’F1 score e la matrice di confusione, sia per valutare quale classificatore fosse di volta in volta ‘migliore’, sia addestrando lo stesso modello su diversi training set.

Abbiamo raggiunto accuracy: 0,880230326295.

Immagine che contiene testo

Descrizione generata automaticamente

Nota: il commento ‘oversampling’ sta ad indicare che non sono stati eseguiti gli step successivi di anomaly detection e undersampling.