PROGETTO MOBD

Petrolo Melissa
0286160

L'obiettivo di questo progetto è quello di risolvere un problema di classificazione binaria in modo tale da ottenere una migliore accuratezza.

Il progetto è stato realizzato sulla piattaforma di Google Colab, con l'utilizzo delle seguenti librerie: Pandas, Numpy e Sklearn già presenti sull'ambiente di lavoro.

1.1 DATA PREPARATION

Prima di eseguire la fase di pre-processing sono state fatte delle analisi sul dataset. Il dataset ha una dimensione di 32562 righe e di 15 colonne denominate: F0, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13, Y; le prime 14 colonne contengono valori numerici o categorici, mentre l'ultima determina la classe di appartenenza. Solo tre colonne contengono valori nulli:

- F1: 1836
- F6: 1843
- F13: 583

Rispetto alla dimensione del dataset i valori nulli non sono significativi e per questo motivo si è deciso di ricorrere alla loro sostituzione anziché all'eliminazione.

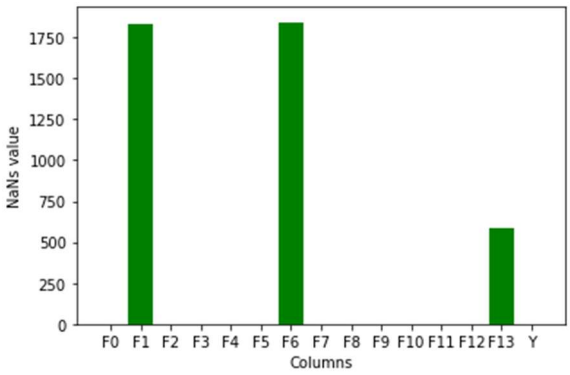

Figura 1 - Valori nulli

Esistono colonne che contengono valori categorici: F1, F3, F5, F6, F7, F8, F9, F13. Nelle colonne F1, F3, F5, F6, F7, F9 non è stato possibile effettuare delle analisi perché non sono stati trovati dei raggruppamenti adeguati al fine di migliorare l'accuratezza del classificatore; invece, nella colonna F8 sono state fatte delle sostituzioni ma non sono stati rilevati cambiamenti significativi durante la fase di classificazione e per questo motivo sono state riconsiderate tutte e cinque le categorie iniziali [caucassian, asian, black, american, other].

Nella colonna F13 inizialmente i singoli stati sono stati raggruppati in base al loro continente di appartenenza. Successivamente la suddivisione è avvenuta in diversi modi:

- Considerando 5 categorie: [europe, asia, north_america, south_america, central_america]
- Considerando 4 categorie: [europe, asia, north_america, south_america]
- Considerando 3 categorie: [europe, asia, america]
- Considerando 2 categorie: [america, no_america]

Non ci sono state grandi differenze nonostante la categoria del north_america, la quale, comprende la maggior parte del dataset. La scelta di utilizzare le cinque categorie [europe, asia, north_america, south_america, central_america] ha permesso di avere un'accuratezza leggermente migliore.

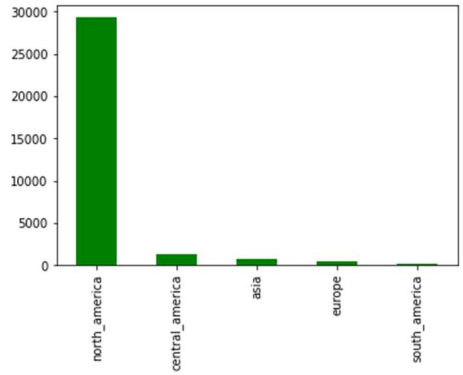

Figura 2 - Valori della colonna F13

1.1.1 Training ser e Test set

Il dataset è stato suddiviso in training set e test set, in modo tale che tutte le scelte effettuate per addestrare il modello vengano fatte solamente sul training set e la fase di valutazione sul testing set.

Lo split è stato fatto considerando un rapporto 80:20.

1.2 PREPROCESSING

La fase di preprocessing è necessaria poiché nel dataset sono presenti valori nulli e colonne categoriche.

Le tecniche utilizzate per la gestione dei valori mancanti, feature scaling e OneHotEncoder sono state eseguite attraverso una Pipeline consentendo di eseguire una serie di azioni diverse per i valori categorici e numerici.

1.2.1 Gestione valori mancanti

Dall'analisi effettuata i valori mancanti sono presenti solo sulle colonne categoriche, ma non conoscendo il dataset del testing set viene effettuata anche sui valori numerici poiché questa procedura viene serializzata per poter essere utilizzata nella fase di testing.

La tecnica utilizzata è il SimpleImputer() che con una strategia 'most_frequent' sostituisce i valori nulli con i valori categorici che sono più frequenti nel dataset, mentre per i valori numerici si è utilizzata una strategia 'mean'.

1.2.2 Feature Scaling

La feature scaling permette di normalizzare il range dei valori delle feature sulla stessa scala e viene applicato solo sui valori numerici.

Non si sono notate grandi differenze tra le due diverse tecniche StardardScaler() e RobustScaler(), ma è stata utilizzata quest'ultima per rendere il dataset più robusto ai possibili outliers.

1.2.3 OneHotEncoder

Data la presenza di colonne categoriche, non è possibile addestrare il modello su valori che non sono numerici, infatti viene applicata la tecnica OneHotEncoder. Vengono create diverse colonne a cui viene assegnato il valore [0,1].

1.2.4 Bilanciamento del dataset

Nel nostro caso il dataset non è bilanciato, si ha [0.7592, 0,2408]; possiamo notare una classe maggioritaria sui valori che assumono 0.

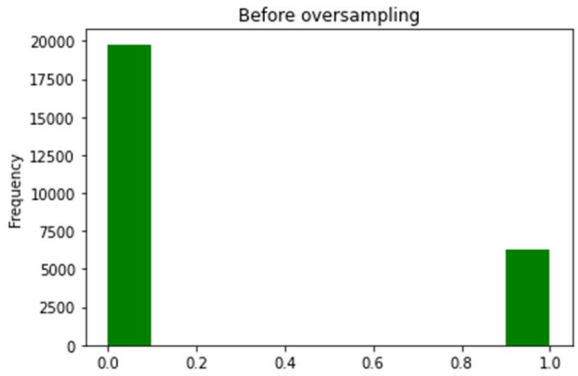

Figura 3- Bilanciamento prima dell'oversampling

Per questo motivo, la scelta sulla tecnica di bilanciamento è stata fatta sulle tecniche di oversampling piuttosto che undersampling, perché utilizzando quest'ultima vengono eliminati più la metà dei dati per ottenere un bilanciamento. Utilizzando invece la tecnica di oversampling, in questo caso KMeansSmote, nonostante possa aggiungere più rumore al dataset con l'aggiunta di molti dati sintetici, è stata la soluzione migliore.

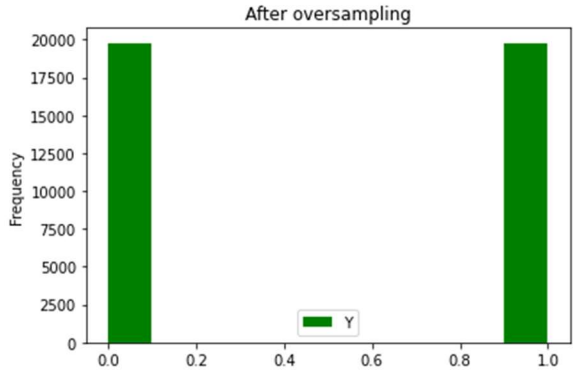

Figura 4 - Bilanciamento dopo l'oversampling

1.2.5 Anomaly detection

Visto che sono stati aggiunti molti dati con l'oversampling, è necessario eseguire anomaly detection per verificare la presenza di anomalie che potrebbero peggiorare le prestazioni. Viene utilizzata la tecnica di Isolation Forest che isola le anomalie attraverso l'utilizzo di alberi di decisione. Considerato il dataset molto grande le anomalie pervenute risultano molto basse con un valore pari a 281, la quale verranno eliminate.

1.3 CLASSIFICATORI

Per ottenere una migliore accuratezza è stata utilizzata la grid search cross su diversi classificatori e parametri. I classificatori utilizzati sono: GradientBoostingClassifier, AdaBoostClassifier, RandomForestClassifier, SVC, DecisionTreeClassifier.

1.3.1 GradientBoostingClassifier

Parametri utilizzati:

- n_estimators = [100, 250, 1000]
- learning_rate = [0.1, 0.5, 1.0]
- max_depth = [2, 3, 5]

Accuratezza migliore: 0,915518 utilizzando come parametri {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100}.

1.3.2 AdaBoostClassifier

Parametri utilizzati:

- base_estimator = DecisionTreeClassifier()

- n_estimators = [100, 250, 1000]
- learning_rate = [0.1, 0.5, 1.0]
- max_depth = [2, 3, 5]

Accuratezza migliore: 0,914830 utilizzando come parametri {'base_estimator__max_depth': 2, 'learning_rate': 0.5, 'n_estimators': 100}.

1.3.3 RandomForestClassifier

Parametri utilizzati:

- n_estimators = [100, 250, 1000]
- max_depth = [2, 5, 10]

Accuratezza migliore: 0,905480 utilizzando come parametri {'max_depth': 10, 'n_estimators': 100}.

1.3.4 SVC

Parametri utilizzati:

- kernel = [‘rbf’, ‘sigmoid’]
- C = [0.1, 1, 10]
- Gamma = [‘scale’]

Accuratezza migliore: 0,871055 utilizzando come parametri {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}.

1.3.5 DecisionTreeClassifier

Parametri utilizzati:

- criterion = [‘gini’, ‘entropy’]
- max_depth = [2, 5, 10, 30]
- splitter = [‘best’, ‘random’]

Accuratezza migliore: 0,905964 utilizzando come parametri {criterion: ‘gini’, 'max_depth': 10, splitter = ‘best’}.

1.4 CONCLUSIONI

I migliori classificatori che consentono di ottenere una maggiore accuratezza sono il GradientBoostingClassifier e AdaBoostClassifier, mentre il risultato peggiore è stato ottenuto con SVC. I classificatori GradientBoostingClassifier e AdaBoostClassifier hanno ottenuto un’accuratezza molto simile, per questo motivo sono stati presi in considerazione anche i valori della matrice di confusione e il risultato ottenuto indica una miglior performance riguardo il primo classificatore. Di seguito i seguenti risultati:

- Training set: Accuratezza 0,920652 - F1: 0,919103
- Testing set: Accuratezza: 0,87502- F1: 0,721233

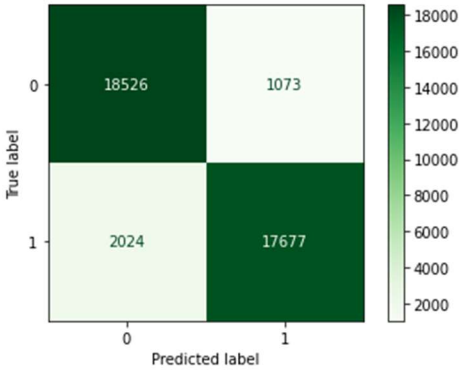

Figura 5- Training set

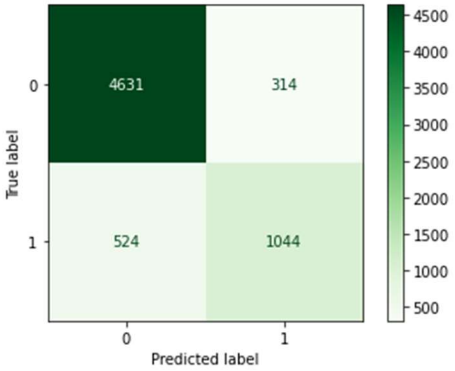

Figura 6- Testing set