Progetto di Big data e business intelligence

Corso a cura dei prof. G.Lombardo, M.Tomaioulo, M.Mordonini

UNIVERSITÀ
DI PARMA

Realizzato da Giuseppe Ricciardi, matricola 306330

Università degli studi di Parma, marzo 2022

Contents

1 Introduzione: looking at the big picture

Lo scopo di questo report è quello di illustrare ed analizzare il processo di realizzazione del progetto per il corso di *Big Data e business intelligence*.
Nello specifico verranno esaminati i vari strumenti e le tecniche utilizzate per le parte di **data analysis** e **data visualization**, inoltre saranno approfondite tecniche di **Pre-processing**: dalla feature selection tramite *informazione mutua* alla dimensionality reduction. Il punto principale di questo report è costituito dall'esposizione degli algoritmi di learning: iniziando con l'analisi della rete neurale fino ad arrivare alla scelta dell'algoritmo di gradient boosting giustificandone la scelta e mostrandone una possibile ottimizzazione atraverso metodi distinti di model tuning.
Il dataset *diabetes_binary_5050split_health_indicators* è costituito da 70692 sondaggi telefonici riguardanti vari aspetti della vita di pazienti medici, ognuna delle 21 features corrisponde ad una domanda fatta ad un paziente. Nel file csv in cui sono raccolte tutte le risposte è inoltre presente una label *Diabetes_binary*, composta da due classi: 0 se il paziente non ha il diabete ed 1 se il paziente ha una forma di prediabete o diabete.
Ci troviamo di fronte quindi ad un problema di **supervised learning** vista la presenza di una label, nello specifico bisogna affrontare un problema di **classificazione binaria**: il programma dovrà essere in grado di riconoscere se un paziente con determinate caratteristiche è affetto da diabete opppure no. Il dataset è bilanciato: è quindi possibile utilizzare come metrica di valutazione della performance l'**accuratezza** (o accuracy). Il dataset è diviso in *training_set* (70% del dataset) e *test_set* (30% restante) che verrà utilizzato esclusivamente sul testing finale, inoltre un 10% del *training_set* viene campionato per generare il *validation_set*, utilizzato per il *model comparison*.
Il programma sviluppato per risolvere questa task è stato scritto interamente in linguaggio python, per quanto riguarda la data analysis e la data visualization sono state utilizzate le librerie *numpy,pandas e matplotlib*, per l'utilizzo della rete neurale sono state implementate le librerie *Keras e Tensorflow*. Per ultima, ma non per importanza, la libreria *scikit* comprende tutti gli algoritmi di learning, eccetto l'algoritmo *xboosting*, e le varie tecniche di model tuning (e relative metriche) e feature selection.

2 Uno sguardo da vicino: data analysis, data visualization e dimensionality reduction

Prima parte fondamentale del programma si occupa dell'analisi dei dati e di una loro rappresentazione grafica.La funzione **data_analysis()** si occupa di tutte le operazioni necessarie. L'utente può scegliere se ottenere un *report generale* digitando 1 oppure un *report dettagliato* con 0.
N.B alla funzione data_analysis() viene passato solo il *training_set*.

2.1 Report generale

In rosso è segnata la target label y *Diabetes_binary*, le **3 features numeriche** vengono rappresentate da un *istogramma* mentre le **18 features categoriche** sono visualizzate attraverso un *grafico a barre*.
Per quanto riguarda le features categoriche è possibile notare la presenza di una forma d'encoding: nella maggior parte dei casi abbiamo dei valori binari 0 ed 1, mentre per una minoranza è presente una *scala ordinale*. E' facilmente visibile che i valori che possono assumere le 3 variabili numeriche sono limitati, per questo motivo non occorre svolgere nessun tipo di operazione di *feature scaling*. Nel dataset non sono presenti missing values e non esiste alcun tipo di rumore. Il report generale è un oggetto *subplot* della libreria matplotlib, consiste in una matrice 6x4 in cui in ogni casella viene creato un istogramma o un grafico a barre in base al tipo di dato. Il controllo per verificare se il dato è di tipo categorico o numerico viene fatto a priori: per sapere se un dato è di tipo categorico il programma controlla che ogni nome della colonna del dataset(ovvero il nome della feature) non sia presente all'interno di un array inizializzato con tutti i nomi delle features numeriche.

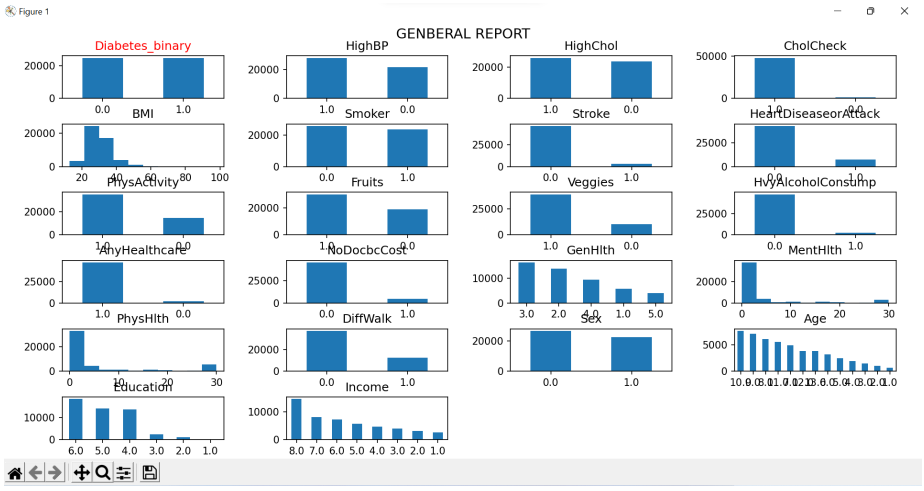

Figure 1: Screenshot del general report

2.2 Report dettagliato

Analizziamo singolarmente le varie features del nostro database utilizzando il *report dettagliato* che oltre alla visualizzazione grafica dei dati ne calcola media e deviazione standard se si tratta di un *dato numerico* oppure mediana e moda nel caso di *dati categorici*

- **HighBP**: è un *dato categorico nominale binario*, 0 rappresenta un paziente che ha un livello di pressione sanguigna basso ed 1 un paziente che ha un livello di pressione sanguigna alta. La sua mediana è 1.0 e la sua moda è 1.

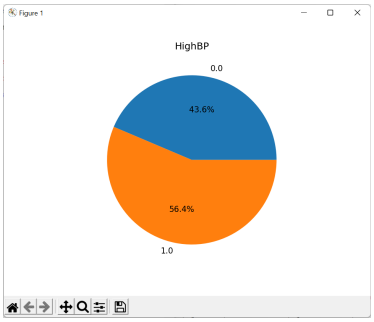

Figure 2: Esempio di data visualization di un dato categorico del detailed report

- **BMI** è un *dato numerico continuo*, rappresenta l'indice di massa corporea del paziente. La sua media è 29.873 mentre la sua deviazione standard è 7.151

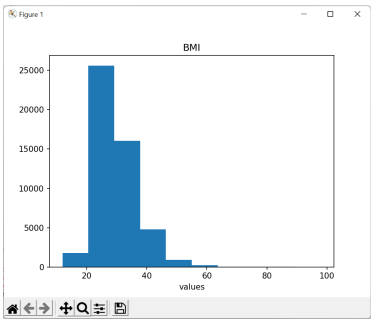

Figure 3: Esempio di data visualization di un dato numerico del detailed report

- **HighCol**: è un *dato categorico nominale binario*, 0 rappresenta un paziente con un tasso di colesterolo basso mentre 1 un paziente con un tasso di colesterolo alto. La sua mediana è 1 e la sua moda è 1.

- **CholCheck** è un *dato categorico nominale binario*, 0 rappresenta un paziente che non ha effettuato nessun controllo del colesterolo negli ultimi 5 anni mentre 1 rappresenta un paziente che ha effettuato almeno un controllo del colesterolo negli ultimi 5 anni. La sua mediana è 1 ela sua moda è 1.

- **Smoker** è un *dato categorico nominale binario*: 0 rappresenta un paziente che non ha fumato 100 pacchetti di sigarette, 1 rappresenta un paziente che ha fumato 100 pacchetti di sigarette. La sua mediana è 0 e la sua moda è 0.

- **Stroke** è un *dato categorico nominale binario*: 0 rappresenta un paziente che non ha mai avuto un infarto, 1 rappresenta un paziente che ha avuto 1 infarto di sigarette. La sua mediana è 0 e la sua moda è 0.

- **HeartdiseasorAttack** è un *dato categorico nominale binario*: 0 rappresenta un paziente che non presenta malattie cardiovascolari mentre 1 rappresenta un paziente con malattie cardiovascolari. La sua mediana è 0 e la sua moda è 0.

- **PhysActivity** è un *dato categorico nominale binario*: 0 rappresenta un paziente che non ha svolto attività fisica negli ultimi 30 giorni mentre 1 rappresenta un paziente che ha svolto attività fisica, la sua mediana è 1 e la sua moda è 1

- **Fruits** è un *dato categorico nominale binario*: 0 rappresenta un paziente che non consuma frutta mentre 1 rappresenta un paziente che consuma frutta almeno 1 volta al giorno. La sua mediana è 1 e la sua moda è 1.

- **Veggies** è un *dato categorico nominale binario*: 0 rappresenta un paziente che non consuma verdura mentre 1 rappresenta un paziente che consuma verdura almeno 1 volta al giorno. La sua mediana è 1 e la sua moda è 1

- **HvyAlcoholConsump** è un *dato categorico nominale binario*: 0 rappresenta un paziente che non consuma molto alcool in una settimana, 1 rappresenta un paziente che consuma più di 14 drink per settimana. La sua mediana è 0, la moda è 0.

- **AnyHealthcare** è un *dato categorico nominale binario*: 0 rappresenta un paziente che non ha assicurazione sanitaria, 1 rappresenta un paziente con assicurazione sanitaria. La mediana è 1, la moda è 1.

- **NoDocbcCost** è un *dato categorico nominale binario*: 0 rappresenta un paziente che nell'ultimo anno non ha potuto permettersi una visita medica per via del prezzo, 1 rappresenta un paziente che ha potuto permettersi una visita medica nell'ultimo anno. La sua mediana è 1, la sua moda è 1.

- **GenHlth** è un *dato categorico ordinale*, rappresenta su una scala da 1 a 5 il livello di salute generale del paziente. La mediana è 3, la moda 5.

- **MentHlth** è un *dato numerico continuo* indica il numero di giorni (da 1 a 30) con poca salute mentale. La media è 3.731, la deviazione standard 8.108
- **PhysHlth** è un *dato numerico continuo* indica il numero di giorni (da 1 a 30) con poca salute fisica. la media è 5.18, la deviazione standard 10.054
- **DiffWalk** è un *dato categorico nominale binario*: 1 indica se il paziente ha difficoltà a salire le scale, 0 se non ha difficoltà. La mediana è 0, la moda è 0.
- **Sex** è un *dato categorico nominale binario*: 0 se il paziente è femmina, 1 maschio. La mediana è 0, la moda è 0.
- **Age** è un *dato categorico ordinale*, indica la fascia d'età del paziente da 1 a 13, la sua mediana è 9 e la moda 10.
- **Education** è un *dato categorico ordinale*, indica su una scala da da 1 a 5 il livello d'istruzione del paziente. La mediana è 5, la moda 6
- **Income** è un *dato categorico ordinale* indica su una scala da 1 ad 8 il livello economico del paziente, mediana è 6, moda 8.

2.3 Dimensionality reduction

In questa sezione ci occupiamo di risolvere il problema legato alla **maledizione della dimensionalità**, avendo a che fare con un dataset composto da 21 features (esclusa la label y) diventa difficile poter rappresentare il dataset in 2 o 3 dimensioni. Per permettere una rappresentazione in 3 dimensioni, il programma utilizza la funzione **dimensionality_reduction()** che implementa due algoritmi dalla libreria *sklearn*: **PCA** e **TSNE**. Per rendere i risultati facilmente visibili e soprattutto per avere tempi d'esecuzione più veloci viene passato il *validation_set*. La funzione crea due copie del set X, una per il TSNE e l'altra per il PCA, e dopo aver eseguito le relative funzioni il programma genererà il seguente plot 3D: in giallo i pazienti senza diabete mentre in verde quelli con diabete.

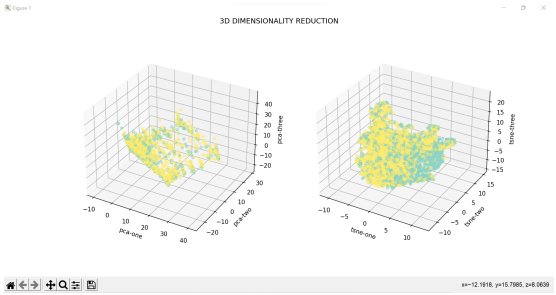

Figure 4: a sinistra risultato dell'algoritmo PCA, a destra quello del TSNE

L'algoritmo PCA cerca l'iperpiano che preserva il più possibile la *varianza* mentre l'algoritmo T-SNE cerca di conservare la *distanza tra i punti* in più dimensioni anche nel grafico con meno dimensioni. La scelta di creare delle copie dei dataset utilizzati per questi due algoritmi deriva dal fatto che entrambi gli algoritmi sono *unsupervisioned*: essi non tengono conto della label riferimento y quando svolgono le loro operazioni di riduzione e di conseguenza i risultati non sono attendibili ai fini del MACHINE LEARNING.

3 Preprocessing: correlation matrix e feature selection

Dopo aver analizzato e visualizzato accuratamente il dataset bisogna occuparsi del preprocessing per cercare di ottenere risultati migliori (o almeno equivalenti) con un minor numero di features, e quindi meno dati da elaborare.

3.1 Correlation matrix

Prima di procedere alla *feature selection* il programma genera una **correlation matrix** tramite l'apposita funzione di pandas e successivamente genera un plot con la funzione *matshow* di matplotlib e infine la esporta su un file excel *correlations.xlsx* utilizzando la funzione *to_excel()* (è richiesto il modulo *openpyxl*).

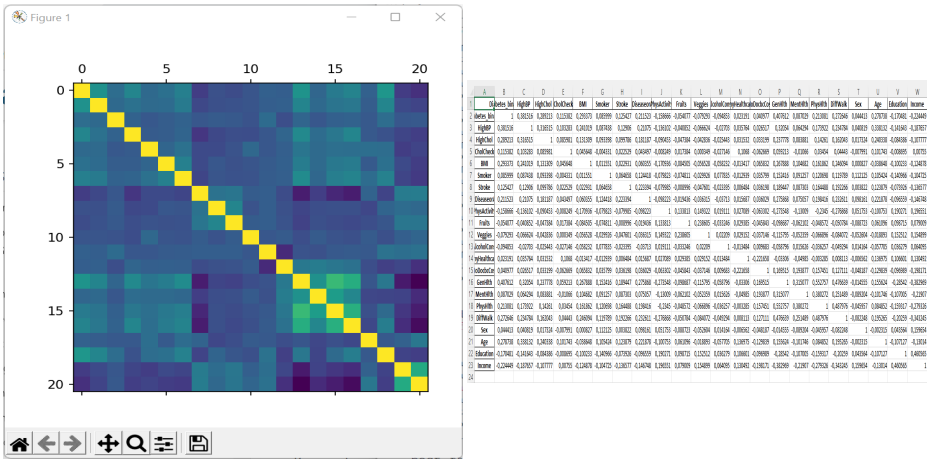

Figure 5: A sinistra il plot della correlation matrix mentre a destra la correlation matrix con i vari valori su excel

Su questo file è possibile valutare le possibili **correlazioni** tra le varie features, bisogna però ricordare che anche nel caso di valori di correlazione alti tra due features non è detto che ci sia un legame causale tra di esse, in questo caso non ci sono stati valori molto alti da essere considerati.

3.2 Feature selection

La funzione **feature_selection()** calcola le 7 migliori features attraverso il metodo *SelectKmodel* della libreria *Sklearn*, la metrica utilizzata per la feature selection è l'**informazione mutua**, che assume valore 0 nel caso le variabili fossero indipendenti ed 1 se le variabili sono estremamente dipendenti: dopo aver selezionato i 7 valori più alti verranno eliminati le restanti features da *X_training*, la funzione effettua un plotting del risultato e ritorna come valori un array che contiene i nomi delle features selezionate (che servirà successivamente nella fase di testing finale).

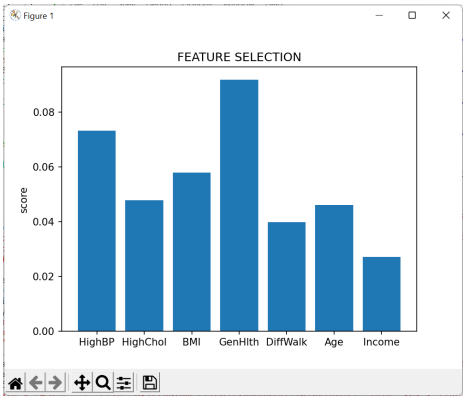

Figure 6: Feature selection plot

Dopo aver provato varie volte le funzioni **neural_network()** e **model_comparison()** è stato possibile verificare che i valori di *accuracy* non variano molto rispetto all'utilizzo di tutte le 21 features, ciò significa che il processo di estrazione delle features più importanti per predire la label funziona correttamente.

4 Una possibile soluzione: uso della rete neurale

L'utilizzo della *rete neurale* avviene attraverso la funzione **neural_network()**. Il programma crea 2 modelli differenti di rete neurali:

1. **Rete neurale senza hidden layer**, il modello è composto soltanto da un *input layer* che ha 7 neuroni (pari al numero di features in ingresso) e come funzione d'attivazione la relu. L'*output layer* è costituito da un neurone con funzione d'attivazione la sigmoide (perchè la task è una classificazione binaria). Il modello viene addestrato con un periodo di 10 epoche, il numero basso è stato scelto per evitare il fenomeno di *overfitting* oltre ad evitare di allungare esponenzialmente i tempi d'esecuzione del programma.

Come già detto in precedenza, la metrica utilizzata per valutare la performance è l'**accuratezza**, dopo diversi test si è ricavato un *valore medio* di 0.744,i picchi presenti nei i grafici sono dovuti alla *stocasticità* della divisione del training e validation set.

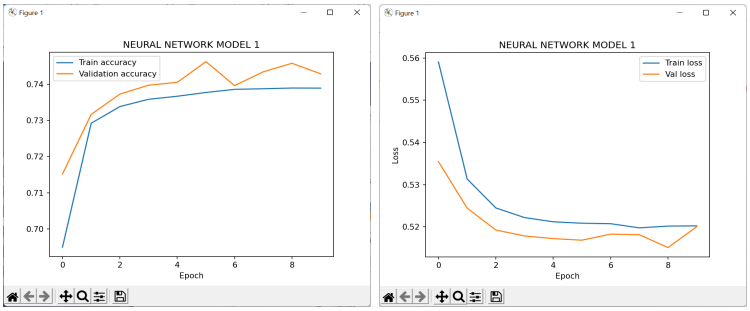

Figure 7: A sinistra la learning curve tra accuracy e numero di epoche mentre a destra tra loss e numero di epoche

2. **Rete neurale con hidden layer**: oltre all'input layer e l'output layer descritti in precedenza in questo modello è presente anche un *hidden layer* con un 5 neuroni, l'introduzione di questo layer serve a sfruttare l'**embedding** di una rete neurale. I risultati sono leggermente inferiori al primo modello,questo perchè non ci sono abbastanza dati per addestrare correttamente una rete più complessa. L'*accuracy* media è di 0.742

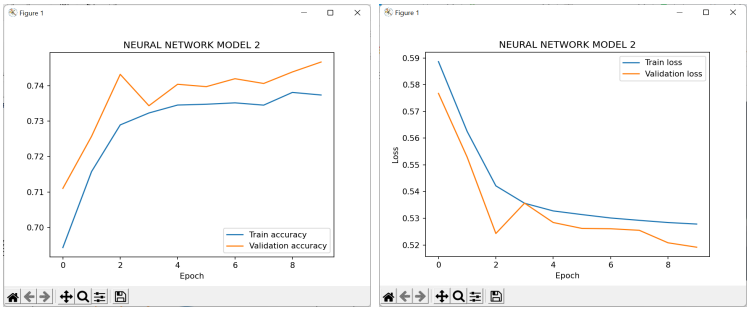

Figure 8: A sinistra la learning curve tra accuracy e numero di epoche mentre a destra tra loss e numero di epoche

Nel programma è presente anche un terzo modello identico al modello 2 ma con un'ottimizzazione Adam, con un learning rate più piccolo ma i risultati mostrano un andamento della learning curve scorretto nonostante un leggero miglioramento dell'accuratezza questo comportamento è molto probabilmente legato ad un fenomeno di *underfitting*.

5 L'importanza del model comparison e sua implementazione

Dopo aver visualizzato i dati, rimosso le feature meno utili per predire la label e aver introdotto una possibile soluzione al problema attraverso la rete neurale, il programma esegue la funzione **model_comparison()** che esegue con parametri default i seguenti algoritmi di classification learning visti a lezione e ne calcola l'accuratezza (**N.B** il testing in questa funzione viene effettuato sul validation-set, ed i valori dell'*accuracy* riportati sono mediati dopo 10 esecuzioni):

- **Regressione logistica**: accuracy = 0,742
- **Decision Tree classifier** accuracy = 0,676
- **Random forest** accuracy = 0,702
- **Gradient boosting** accuracy = 0,750
- **Xboosting** accuracy = 0,746

```
Executing classification algorithms with standard parameters...
Logistic regression accuracy: 0.747
Decision Tree Classifier accuracy: 0.68
Random forest accuracy: 0.711
Gradient boosting accuracy: 0.752
Xboost accuracy: 0.751
```

Figure 9: Screenshot da console dell'esecuzione di model_comparison()

Si può evincere facilmente che il modello scelto per il model tuning e il testing finale è il **gradient boosting**

6 Model tuning e testing finale

Una volta scelto l'algoritmo da utilizzare si può procedere con il model tuning attraverso la funzione **gradient_model_tuning()**, essa contiene l'implementazione di due algoritmi: l'*Halving random search* e il *Bayes Search*. Gli hyperparametri che verranno ottimizzati sono: *learning_rate,n_estimators e max_depth*.

6.1 Halving random search

L'**Halving random search** è implementato attraverso la libreria *sklearn*. La ricerca iniziale dei parametri avviene con un piccolo numero di esempi e iterativamente vengono scelti gli hyperparameters migliori. Può essere vista come una competizione tra i possibili candidati, che rappresentano le combinazioni degli hyperparametri selezionati in maniera casuale, passati alla funzione sotto forma di dizionario. Ogni turno della competizione corrisponde ad un'iterazione: con il passare delle iterazioni aumenta il numero di risorse.

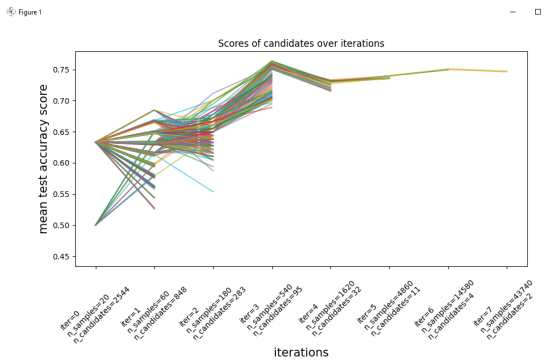

Figure 10: "iter" rappresenta l'iterazione,"n_samples" rappresenta il numero di samples usati in quell'iterazione e "n_candidates" rappresenta il numero di candidati in quella iterazione.

Il candidato migliore è quello che "sopravvive" a tutte le iterazioni.

6.2 Bayes Search

Un altro approccio all'ottimizzazione del gradient boosting consiste nell'algoritmo **BayesSearchCV** che sfrutta il principio di ottimizzazione baessiana: trovare il minimo di una *funzione obiettivo* usando un processo Gaussiano, la funzione obiettivo consiste nel trovare il miglior modello d'output avendo in input i parametri del modello (sotto forma di dizionari). Il vantaggio di questo algoritmo è che esplora automaticamente le regioni dello spazio delle variabili da valutare più promettenti scartando le peggiori in maniera automatica, incidendo sul tempo d'esecuzione. Per questa ragione oltre agli hyperparameters citati prima verrà anche modificato il parametro *min_samples_leaf*.

Per la rappresentazione grafica di questo metodo viene chiamato il metodo **plot_objective()** della libreria *scikit-optimize* che restituisce un grafico sulle *dipendenze parziali*: mostra l'influenza di ogni spazio di ricerca degli hyperparameters sulla funzione obiettivo.

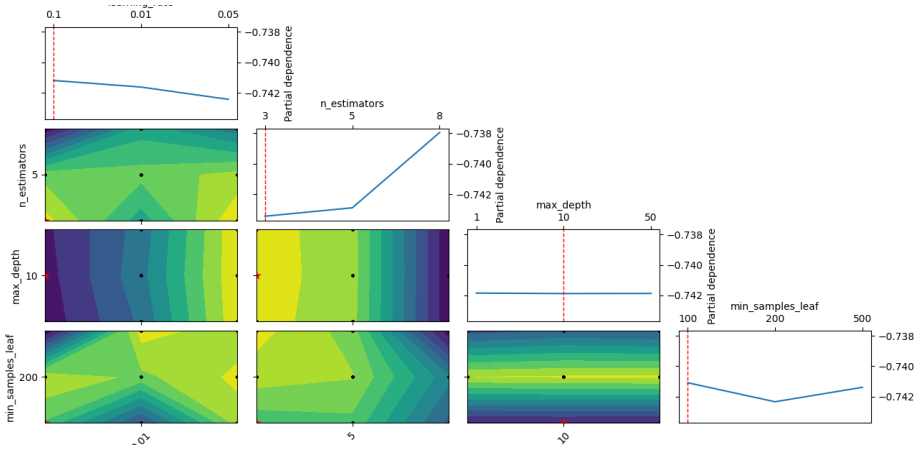

Figure 11: Dependence plot

La *diagonale* di questa matrice mostra l'effetto di una singola dimensione sulla funzione obiettivo, mentre gli altri grafici mostrano l'effetto della variazione delle due dimensioni, relative ai parametri delle righe e delle colonne, sulla funzione obiettivo. La **dipendenza parziale** viene calcolata facendo la media del valore della funzione obiettivo per un numero di campioni casuali nello spazio di ricerca, mantenendo una o due dimensioni fisse a intervalli regolari. Questo calcola l'effetto della variazione delle altre dimensioni e mostra l'influenza di una o due dimensioni sulla funzione obiettivo. Vengono anche mostrati piccoli *punti neri* per i punti che sono stati campionati durante l'ottimizzazione. Una *stella rossa* indica il minimo meglio osservato.
N.B: "The Partial Dependence plot is only an estimation of the surrogate model which in turn is only an estimation of the true objective function that has been optimized. This means the plots show an "estimate of an estimate" and may therefore be quite imprecise" (fonte: *documentazione scikit-optimize riguardo la funzione plot_objective()*)

6.3 Testing finale

Dopo aver memorizzato gli hyperparameters ottimali il programma addestra un modello gradient boosting ed effettua il test finale sul **test_set**.Il programma effettua anche il plotting della **confusion matrix** dei risultati sul test_set, la matrice riporta il numero di:

- positivi predetti correttamente (true positive), in basso a destra
- positivi predetti come negativi (false negative), in basso a sinistra
- negativi predetti come positivi (false positive), in alto a destra
- negativi predetti come negativi (true negative), in alto a sinistra

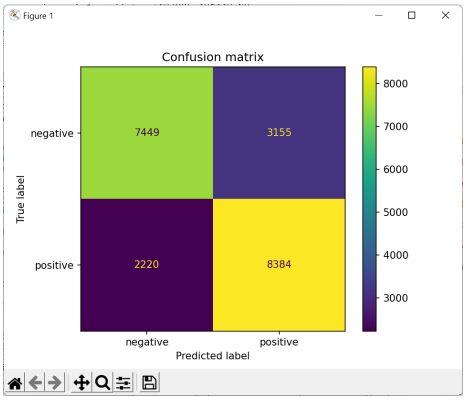

Figure 12: Confusion matrix

I valori finali dell'*accuratezza* si aggirano intorno al 0.75 che per il nostro scopo risulta un buon punto d'arrivo.