
Università degli studi di Roma Tor Vergata

Facoltà di Ingegneria Informatica


# PROGETTO MOBD

Breve relazione e manuale d'uso


Anno Accademico
2019/2020

Lasco Giuseppe 0286045
Marcucci Marco 0286352

# Sommario

# INTRODUZIONE

Il progetto consiste nell'ottenere un modello di machine learning che, attraverso un'adeguata configurazione, permetta di ottenere il miglior risultato, rispetto alla metrica 'F1_MACRO', relativo ad un problema di classificazione multi-classe a 4 classi.
Il dataset fornito per addestrare il modello è costituito da 8000 istanze e 20 features.

# PROGETTAZIONE

Per il raggiungimento dell'obiettivo, il dataset fornitoci viene diviso in una parte di training costituita dall'80% delle istanze (6400 record) e una parte di testing contenente il restante 20% (1600 record).
La fase di progettazione si compone di due macro-fasi:

- Pre-processamento
- Tuning degli iper-parametri

## Pre-processamento

Nel machine learning la pre-elaborazione (o pre-processing) è la fase in cui i dati vengono preparati ed analizzati, prima di avviare l'algoritmo di apprendimento.

Questa fase si snoda in ulteriori step:

- Gestione valori nulli;
- Gestione degli outliers;
- Scaling dei dati;
- Features Selection;
- Balancing.

## Gestione dei valori nulli

Si è notato che il dataset presenta delle istanze con dei valori nulli che potrebbero compromettere la fase di apprendimento; tali valori vengono gestiti sostituendoli con la media dell'attributo ricavata solo dal training set.

Prima di questo step, il dataset evidenziava la presenza di valori mancanti nelle seguenti proporzioni:

| F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | F10 |
|---|---|---|---|---|---|---|---|---|---|
| 6 | 6 | 1 | 1 | 4 | 6 | 4 | 9 | 6 | 6 |

| F11 | F12 | F13 | F14 | F15 | F16 | F17 | F18 | F19 | F20 |
|---|---|---|---|---|---|---|---|---|---|
| 5 | 6 | 3 | 10 | 2 | 6 | 4 | 7 | 5 | 3 |

## Gestione degli outliers

Gli outliers rappresentano tutti quei valori che si distanziano molto dalla media di una distribuzione. Tali valori posso influenzare in maniera negativa l'algoritmo di apprendimento.

Si utilizza la libreria *scipy* che fornisce l'implementazione di *z-score,* un noto indice statistico:

$$z_{score} = \frac{x - \mu}{\sigma}$$

Vengono considerati outlier tutti i valori il cui |*z-score*| è maggiore di 3.

Si è notato che la rimozione degli outlier, prima di effettuare lo scaling dei dati, migliora i risultati del modello in quasi tutte le configurazioni.

## Scaling dei dati

Lo scaling è una tecnica che permette di rendere i dati omogenei e indipendenti dall'unità di misura, in modo da individuare con più facilità le eventuali correlazioni tra i dati a disposizione.

In un primo momento la scelta è ricaduta su *standardScaler* e *MinMaxScaler* che sono comunemente più utilizzati.
In seguito, per ampliare lo spettro di configurazioni possibili, si è scelto di inserire ulteriori tecniche di scaling quali *robustScaler* e *MaxAbsScaler.*
il primo sottrae al dato la mediana e lo scala utilizzando l'Inter Quartile Range, cioè l'intervallo di valori compresi tra il primo e il terzo quantile, mentre il secondo trasforma i valori, per ogni feature, in un intervallo compreso tra -1 e 1.
Si è notato che al variare dei classificatori non vi è uno scaler migliore in assoluto.

## Features selection

La features selection è una tecnica che permette di scartare degli attributi dal dataset che non sono particolarmente rilevanti per l'addestramento del modello, in relazione all'attributo target.

Per l'implementazione della tecnica si è utilizzata la funzione *SelectKBest* della libreria *scikit-learn.*
Tale funzione permette di selezionare le k features più rilevanti utilizzando un indice statistico come discriminante.
Gli indici statistici utilizzati sono:

- *chi2*;*
- *f_classif;*
- *mutual_info_classif.*

Il parametro k si è fatto variare da 10 a 20, dato che con k minori si è evinto un peggioramento del modello in termini di 'F1_SCORE'.
In tutte le configurazioni provate il numero di features ottimale risulta essere 15.

*l'indice statistico *chi2* può essere utilizzato solamente se i valori della feature sono positivi, quindi per utilizzare tale indice è necessario utilizzare esclusivamente lo scaler MinMaxScaler dopo il balancing e prima di features selection.

## Balancing

Il balancing è una tecnica che permette di evitare che il modello finale sia poco addestrato nel riconoscere le classi meno numerose nel training set; nel dataset fornito le percentuali delle classi target sono:

- CLASS 1: 33.675 %
- CLASS 2: 15.9875 %
- CLASS 3: 20.6625 %
- CLASS 4: 29.675 %

Sono state utilizzate diverse tecniche di Undersampling e Oversampling presenti nella libreria *imbalanced-learn*. Nelle prove effettuate i risultati migliori sono stati ottenuti con tecniche di Oversampling, in particolare SMOTE e le sue varianti.

Nella fase di addestramento del modello con l'utilizzo della tecnica di balancing svm_smote il dataset originario passa da $8000$ a $10132$ istanze.

## Considerazioni

Sono state testate diverse permutazioni per il pre-processing dei dati (scaling, features selection, balancing) senza riscontrare netti miglioramenti o peggioramenti; per la configurazione ottimale si è utilizzato il seguente ordine di tecniche:

1. Gestione dei valori nulli;
2. Gestione degli outlier;
3. Scaling;
4. Balancing;
5. Features selection.

# Tuning degli iper-parametri

L'obiettivo di questa fase è quello di ottenere la configurazione migliore unendo le possibili combinazioni di pre-processing con i vari classificatori e i propri iper-parametri.
La funzione utilizzata per il tuning degli iper-parametri è *GridSearchCV* presente nella libreria *scikit-learn*; essa si basa su diversi parametri, tra i quali:

- ***estimator***: rappresenta il classificatore con cui addestrare il modello;
- ***param_grid***: qui vengono specificate le configurazioni degli iper-parametri da passare al classificatore selezionato;
- ***scoring***: rappresenta la metrica sulla quale ottenere il miglior risultato;
- ***cv***: rappresenta il numero di fold con cui effettuare cross-validation.

scelto il classificatore e una griglia di iper-parametri validi, questa funzione addestra il modello considerando tutte le possibili combinazioni degli iper-parametri e ricava la configurazione ottimale, in cross-validation, sulla base della metrica scelta.

In un primo momento si è scelto di testate i classificatori a noi noti tra cui SVM e RandomForest. Questi ultimi, però, non hanno dato esiti interessanti per superare la baseline nonostante le diverse configurazioni sia di pre-processing che di iper-parametri.
I risultati ottenuti sono riassunti nella *Tabella 1*.

| | Svm.SVC | RandomForest |
|---|---|---|
| **F1_SCORE** | 0.8518341990443112 | 0.7901135105199624 |
| **Iper-parametri** | {'C': 400, 'degree': 2, 'gamma': 'scale', 'kernel': 'poly'} | {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 4, 'n_estimators': 1000, 'n_jobs': -1} |
| **Outliers** | without_outliers | without_outliers |
| **Scaling** | robustScale | minMaxScaler |
| **Features Selection** | mutual_info_classif with k=15 | mutual_info_classif with k=15 |
| **Balancing** | smote_tomek | svm_smote |

*Tabella 1*

Successivamente sono stati testati altri classificatori non affrontati durante il corso, quali *svm.NuSVC*, *neural_network.MLPClassifier* e *disciminant_analysis.QuadraticDiscriminantAnalysis*, che hanno dato risultati migliori dei precedenti classificatori, raggiungendo il massimo con QDA, con cui si è scelto di generare il modello finale per la fase di testing.
I risultati ottenuti sono riassunti nella *Tabella 2.*

| | svm.NuSVC | MLPClassifier | QDA |
|---|---|---|---|
| **F1_SCORE** | 0.8459073726397118 | 0.8774504322912312 | 0.893716449373632 |
| **Iper-parametri** | {'class_weight': 'balanced', 'coef0':0 'degree': 2, 'gamma': 'scale', 'kernel': 'rbf'} | {'activation': 'relu', 'hidden_layer_sizes':(100,50), 'learning_rate': 'adaptive', 'learning_rate_init': 0.01, 'max_iter': 1000, 'solver': 'sgd'} | {'reg_param': 0.001, 'store_covariance': True, 'tol': 5e-05} |
| **Outliers** | without_outliers | with_outliers | without_outliers |
| **Scaling** | standardScaler | minMaxScaler | max_abs_scaler |
| **Features Selection** | mutual_info_classif with k=15 | mutual_info_classif with k=15 | mutual_info_classif with k=15 |
| **Balancing** | smote | smote_nc | svm_smote |

*Tabella 2*

Oltre ai classificatori presentati precedentemente, sono stati testati altri modelli tra cui:

- *naive_bayes.GaussianNB;*
- *disciminant_analysis.LinearDiscriminantAnalysis;*
- *AdaBoostClassifier(sklearn.linear_model.Perceptron);*
- *sklearn.neighbors.KNeighborsClassifier;*
- *BaggingClassifier(DecisionTreeClassifier);*
- *gaussian_process.GaussianProcessClassifier*.

Mentre i primi cinque raggiungevano valori di 'F1_MACRO' molto bassi, il sesto classificatore è stato impossibile da testare in quanto provoca saturazione immediata della RAM nonostante il parametro *copy_X_train* fosse impostato al valore FALSE.

# STRUTTURA DEL CODICE E MANUALE D'USO

## Struttura del codice

Il codice si compone di tre parti:

1. Test delle combinazioni possibili di pre-processing e classificatori al fine di ottenere la miglior configurazione rispetto alla metrica 'F1_MACRO';
2. Addestramento e salvataggio, considerando l'intero dataset fornito, della configurazione migliore ottenuta nel punto precedente;
3. Script con cui pre-processare e valutare il testing set.

I moduli *scaler.py, balancer.py e classifier.py* contengono rispettivamente tutti gli scaler, tipi di balancing e classificatori utilizzati.

Il modulo *hyperparameter_tuning* contiene l'implementazione descritta nel punto 1.
Il modulo *training_best_configuration* contiene l'implementazione descritta nel punto 2.
Il modulo *eval_testing* contiene l'implementazione descritta nel punto 3.

Nella cartella *html* è disponibile la documentazione del codice; il file *index.html* permette di navigare all'interno della stessa.

## Manuale d'uso

Per valutare la metrica 'F1_MACRO' su un testing set eseguire i seguenti passaggi:

1. Installare le seguenti librerie per il corretto funzionamento dei moduli:
   - *scikit-learn v.0.23.1*
   - *pandas v.1.0.5*
   - *imbalanced-learn v.0.7.0*
   - *scipy v.1.4.1*
   - *numpy v.1.18.5*
   - *joblib v.0.15.1*
2. Inserire il percorso del testing_set (compresa l'estensione '.csv') nella prima riga del file *config.txt*, assicurandosi che sia la prima e l'unica riga del file;
3. Avviare il *main()* del modulo *eval_testing*;
4. Il risultato verrà stampato a schermo.

Nel caso si volesse eseguire il modulo *hyperparameter_tuning*, i risultati delle configurazioni testate saranno disponibili nel file *result.txt*.

# BIBLIOGRAFIA


- https://scikit-learn.org/stable/
- https://pandas.pydata.org/
- https://imbalanced-learn.readthedocs.io/en/stable/
- https://numpy.org/
- https://stackoverflow.com/
- https://www.datasklr.com/select-classification-methods
- https://pdoc3.github.io/pdoc/