Metodi di Ottimizzazione per Big Data

Progetto A.A 2019/2020

Tummolo Gabriele

0283629

Indice:

Introduzione:

Il progetto consiste nel risolvere un problema di classificazione multiclasse, utilizzando un modello per massimizzare la metrica F1 definita dalla seguente formula $F1 = 2 * precision * recall/(precision + recall)$. Per l'addestramento del modello e successivamente per la verifica è stato utilizzato un dataset formato da 8000 istanze con 20 attributi, che è stato suddiviso in training set e test set.

Progettazione:

La fase di progettazione è stata divisa in più parti per ottenere i risultati migliori.

In primo luogo sono stati aggiunti i valori mancanti del dataset, calcolando la media dell'attributo in riferimento al training set. Successivamente è stato effettuato lo scaling dei dati utilizzando diversi metodi, grazie al quale è stato possibile eliminare valori anomali , in particolare sono state utilizzate le funzioni :

```
Scaling: StandardScaler
F1 for SVM : 0.1303223046516476
F1 for MLP : 0.8224893736645691
F1 for DecisionTree : 0.6367764476018005
F1 for RandomForest : 0.7881643506013544
F1 for NaiveBayes : 0.5257009040834627
F1 for KNeighbors : 0.786594933639854
```

Figura 1 Scaling StandarScaler

```
Scaling: Normalizer
F1 for SVM : 0.7675576688733785
F1 for MLP : 0.8363581679792549
F1 for DecisionTree : 0.6269728744044555
F1 for RandomForest : 0.7950454387476822
F1 for NaiveBayes : 0.5253235532538489
F1 for KNeighbors : 0.7799085911652501
```

Figura 1 Scaling Normalizer

```
Scaling: MinMaxScaler
F1 for SVM : 0.8098283458023074
F1 for MLP : 0.8630645925597304
F1 for DecisionTree : 0.6213597311451553
F1 for RandomForest : 0.7887562755531323
F1 for NaiveBayes : 0.5252878062203404
F1 for KNeighbors : 0.7831894619032274
```

Figura 3 Scaling MinMaxScaler

```
Scaling: MaxAbsScaler
F1 for SVM : 0.7593085852892673
F1 for MLP : 0.8495953771818553
F1 for DecisionTree : 0.6122700256768148
F1 for RandomForest : 0.7908852791618745
F1 for NaiveBayes : 0.5306502252519455
F1 for KNeighbors : 0.786684373234539
```

Figura 4 Scaling MaxAbsScaler

```
Scaling: RobustScaler
F1 for SVM : 0.14876646865794865
F1 for MLP : 0.8320026231608625
F1 for DecisionTree : 0.6167292694859235
F1 for RandomForest : 0.7839690701673349
F1 for NaiveBayes : 0.5146877110987318
F1 for KNeighbors : 0.7864944547643199
```

Figura 5 Scaling RobustScaler

```
Scaling: QuantileTransformer
F1 for SVM : 0.7327014837728594
F1 for MLP : 0.8348425807021255
F1 for DecisionTree : 0.597384865016067
F1 for RandomForest : 0.788950136676862
F1 for NaiveBayes : 0.47008107962803813
F1 for KNeighbors : 0.781069062470529
```

Figura 6 Scaling QuantileTransformer

Utilizzando questi diversi tipi di funzioni si può notare in particolare come SVM sia molto sensibile allo scaling in base a il metodo che si utilizza, mentre gli altri classificatori non variano molto. E' stato deciso di procedere con la progettazione utilizzando la funzione MinMaxScaler() che da come si vede nelle figure ha i risultati migliori in particolare per SVM e MLP. Si è proceduto con la fase di Feature Selection, che consiste nell'eliminare attributi riducendo cosi il costo dell'apprendimento; è stato utilizzato un metodo Fliter e successivamente un metodo

Wrapper. Per il metodo Filter, che tenta di valutare i meriti degli attributi dei dati ignorando il classificatore, è stato eseguito il PairPlot del dataset che ha prodotto la seguente immagine:

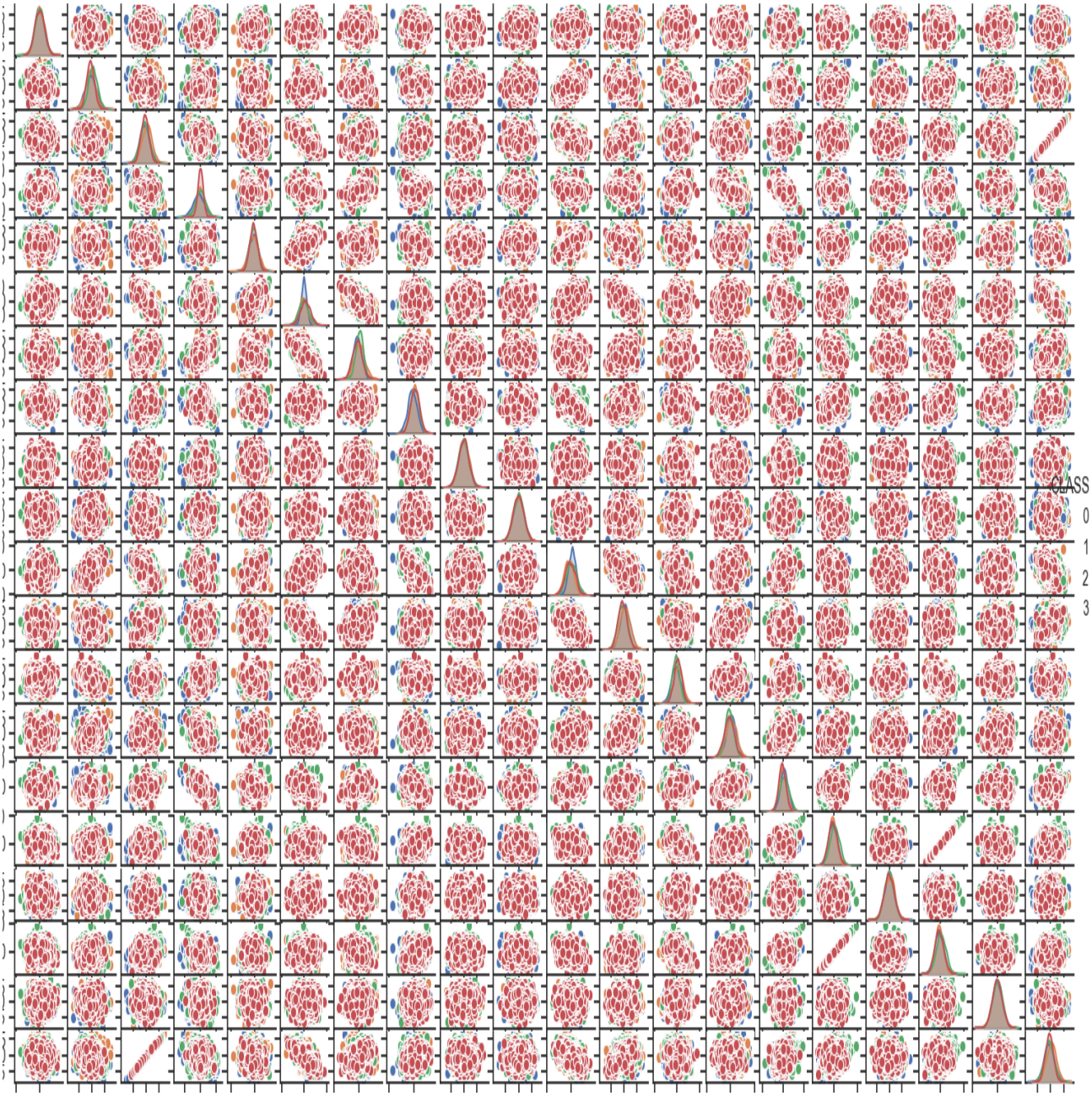

Figura 7 PairPlot Dataset

I grafici sulla diagonale principale rappresentato la distribuzione dei dati in base alla classe di appartenenza e si è notato che gli attributi F19, F9, F10, F1 e F17 , cosi come elencate , hanno un grado di dipendenza minore rispetto all'attributo target. Sono state eliminate da 0 a 5 attributi notando che i risultati migliori sono stati ottenuti nella configurazione con 15 attributi (non sono stati eliminati più attributi in quanto dai grafici si poteva vedere che solo gli attributi

elencati precedentemente avevano un grado di indipendenza rilevante, evitando cosi di eliminare attributi e informazioni utili per la classificazione).

Successivamente è stato utilizzato un metodo Wrapper di Feature Selection che non ha ottenuto risultati migliori o abbastanza soddisfacenti rispetto al metodo Filter. Inoltre è stato effettuato Sampling che ha permesso di bilanciare la percentuale degli elementi delle classi, ottenendo un training set bilanciato; sono stati utilizzati tutti e 3 i tipi di Sampling visti in altri corsi: SMOTE, OverSampling e UnderSampling. OverSampling e UnderSampling sono stati scartati visto che:

- OverSampling duplica elementi già esistenti che nel dataset per equilibrare gli elementi delle classi
- UnderSampling elimina elementi dal dataset portando ad uno stesso livello di elementi tutte le classi, rischiando cosi di eliminare dati rilevanti

inoltre rispetto a SMOTE le prestazioni risultavano peggiori.

Dopo aver separato il dataset in training e test set, è stato effettuato tuning. Il training set è stato diviso in training e validation set, questo è servito per valutare le combinazioni migliori di parametri dei classificatori testando i vari classificatori sul validation set.

I classificatori utilizzati sono:

- **MLP** (MultiLayer perceptron)
- **SVM** (Support Vector Machines)
- **DecisionTreeClassifier**
- **RandomForestClassifier**
- **NaiveBayes**
- **KNeighbors**

MLP:

Per MLP sono stati considerati i seguenti parametri:

```
parameters = [{
    'hidden_layer_sizes': [(100, 50), (100, 50, 25), (200,)],
    'activation': ['identity', 'logistic', 'tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'learning_rate_init': [0.1, 0.01, 1e-3, 1e-4],
    'learning_rate': ['invscaling', 'constant', 'adaptive'],
}]
```

Figura 8 Parametri MLP

- **Hidden_layer_sizes** : definisce il numero di livelli e il numero di percettori per ogni livello, sono stati considerati al massimo 3 livelli con un massimo di 200 percettori , per evitare un elevato tempo computazionale
- **Activation**: definisce le funzioni utilizzate
- **Solver**: definisce i solutori per l'ottimizzazione
- **Learning_rate_int**: definisce la velocita con cui il modello si adatta ai dati
- **Learning_rate**: definisce in che modo il modello di adatterà

SVM:

Per SVM sono stati considerati i seguenti parametri:

```
parameters = [
              {"kernel": ['rbf'], 'C': [0.1, 1, 10, 25, 50, 100],
               "gamma": [10e-4, 10e-3, 10e-2, 10e-1, 10, 10 ** 2, 10 ** 3, 10 ** 4],
               "decision_function_shape": ["ovo", "ovr"]
               },
              {
               "kernel": ['linear'], "C": [0.1, 1, 10], "decision_function_shape": ["ovo", "ovr"]
              }
]
```

Figura 9 Parametri SVM

- **Kernel**: definisce il tipo di kernel che si utilizza per effettuare il kernel trick
- **C**: definisce la tolleranza dell'errore di classificazione
- **Gamma**: definito solo per kernel 'rbf'
- **Decision_function_shape**: utilizzato sia 'ovo' che 'ovr'

DecisionTreeClassifier:

Per DecisionTreeClassifier sono stati considerati i seguenti parametri:

```
param_grid = {
            'criterion': ['entropy', 'gini'],
            'splitter': ['best', 'random'],
            'max_features': [None, 'auto', 'log2'],
            'min_samples_leaf': [1, 5, 10],
            'max_depth': [10, 25, 50, 100],
            'min_samples_split': [2, 7, 15]
}
```

Figura 10 Parametri DecisionTreeClassifier

- **Criterion**: definisce la 'qualità' dello split
- **Splitter**: definisce la strategia utilizzata per effettuare lo split
- **Max_features**: definisce il numero di attributi utilizzati per effettuare lo split migliore

- **Min_samples_leaf**: definisce il numero minimo di campioni richiesto per essere in un nodo foglia
- **Max_depth**: definisce la massima profondità dell'albero
- **Min_samples_split**: definisce il numero minimo di campioni richiesti per dividere un nodo interno

RandomForestClassifier:

Per RandomForestClassifier sono stati considerati i seguenti parametri:

```
param_grid = {
    'criterion': ['entropy', 'gini'],
    'max_features': [None, 'auto', 'log2'],
    'min_samples_leaf': [1, 5, 10],
    'max_depth': [10, 25, 50, 100],
    'min_samples_split': [2, 7, 15],
    'n_estimators': [100, 150, 300, 400]
}
```

Figura 11 Parametri RandomForestClassifier

Rispetto a DecisionTreeClassifier c'è un solo parametro in più:

- **N_estimators**: definisce il numero di alberi della 'foresta'

NaiveBayes:

Per NaiveBayes sono stati considerati i seguenti parametri:

```
parameters = [{
    'priors': [None, [0.25, 0.25, 0.25, 0.25]],
    'var_smoothing': [10e-9, 10e-6, 10e-3, 10e-1]
}]
```

Figura 12 Parametri NaiveBayes

- **Priors**: definisce come sono distribuite in percentuali le classi
- **Var_smoothing**: definisce la varianza che deve essere aggiunta per stabilizzare il calcolo

KNeighbors:

Per KNeighbors sono stati considerati i seguenti parametri:

```
parameters = [{
    'n_neighbors': [5, 7, 10, 15],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [30, 35, 40, 50],
    'p': [1, 2, 3, 4],
}]
```

Figura 13 Parametri KNeighbors

- **N_neighbors**: definisce il numero di 'vicini' da utilizzare per impostare le query
- **Weights**: definisce la funzione 'peso'
- **Algorithm**: definisce il tipo di algoritmo utilizzato
- **Leaf_size**: definisce la dimensione delle foglie
- **P**: definisce il parametro di potenza per la metrica Minkowski

Molti parametri sono stai omessi per ridurre i tempi computazionali.

Conclusione

Per concludere sono stati valutati i vari classificatori con le condizioni migliori in una prima fase calcolate.

MLP:

```
Best parameters:

{'activation': 'relu', 'hidden_layer_sizes': (100, 50), 'learning_rate': 'adaptive', 'learning_rate_init': 0.01, 'solver': 'sgd'}
```

Figura 14 Parametri migliori per MLP

```
F1 for MLP : 0.8671137793404103
```

Figura 15 Risultato finale MLP

SVM:

```
Best parameters:

{'C': 10, 'decision_function_shape': 'ovo', 'gamma': 10, 'kernel': 'rbf'}
```

Figura 16 Parametri migliori per SVM

```
F1 for SVM : 0.8027149285177667
```

Figura 17 Risultato finale SVM

DecisionTreeClassifier:

```
Best parameters:

{'criterion': 'entropy', 'max_depth': 100, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'best'}
```

Figura 18 Parametri migliori DecisionTreeClassifier

```
F1 for DecisionTree : 0.6148049779507877
```

Figura 19 Risultato finale DecisionTreeClassifier

RandomForestClassifier:

```
Best parameters:

{'criterion': 'entropy', 'max_depth': 100, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 400}
```

Figura 20 Parametri migliori RandomForestClassifier

```
F1 for RandomForest : 0.787436288124805
```

Figura 21 Risultato finale RandomForestClassifier

NaiveBayes:

```
Best parameters:

{'priors': None, 'var_smoothing': 1e-09}
```

Figura 22 Parametri migliori NaiveBayes

Figura 23 Risultato finale NaiveBayes

KNeighbors:

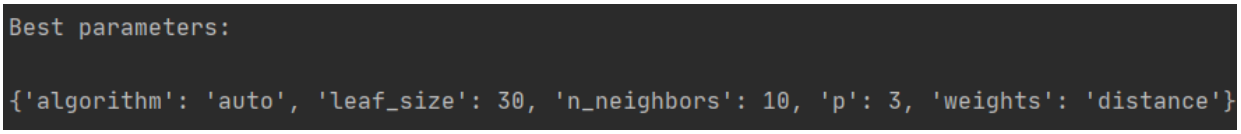

Figura 24 Parametri migliori KNeighbors

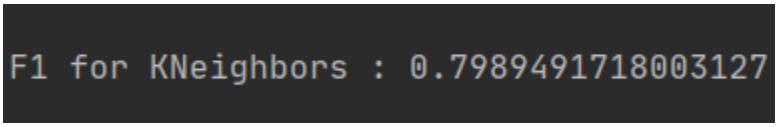

Figura 25 Risultato finale KNeighbors

Come si evince dai risultati finali il classificatore migliore è MLP, si potrebbe migliorare ulteriormente aumentando i livelli o i precettori per ogni livello, ma questo aumenterebbe la complessità computazionale aumentando di conseguenza il tempo di esecuzione.