Metodi di Ottimizzazione per Big Data

Progetto A.A. 2020/2021

0287567

Paolo Melissari

Indice

Introduzione

Lo scopo del progetto è quello di utilizzare modelli di machine learning per risolvere un problema di classificazione multiclasse.

Per la fase di training si è utilizzato un dataset contenente 20 Feature e 8000 record (alcuni dei quali avevano dei dati mancanti).

Per la fase di test verrà utilizzato un dataset contenente 2000 record.

Progettazione

La fase di progettazione è stata divisa in 2 parti:

- Processamento dei dati, composto da scaling, sampling e feature selection
- Scelta dei modelli e strategia per il tuning degli iperparametri

Per quanto riguarda il processamento dei dati, per prima cosa si è diviso il dataset in due parti: training set (contenente l'80% dei dati iniziali, cioè 6400 record ognuno contenente informazioni di 20 feature) e test set (contenente il restante 20%, cioè 1600 record ognuno contenente informazioni di 20 feature).

Poi sono stati considerati tutti i record che avevano dati mancanti per una o più feature, sostituendo a questi ultimi il valore medio che quella feature ha assunto all'interno del dataset.

Successivamente, sul training set è stato effettuato lo scaling in modo da mitigare la sensibilità dei modelli alla scala dei dati. In prima analisi, è stato utilizzato il RobustScaler della libreria sklearn. La scelta è ricaduta su questo scaler perché più resistente agli outlier rispetto allo StandardScaler. Infatti, il RobustScaler 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):

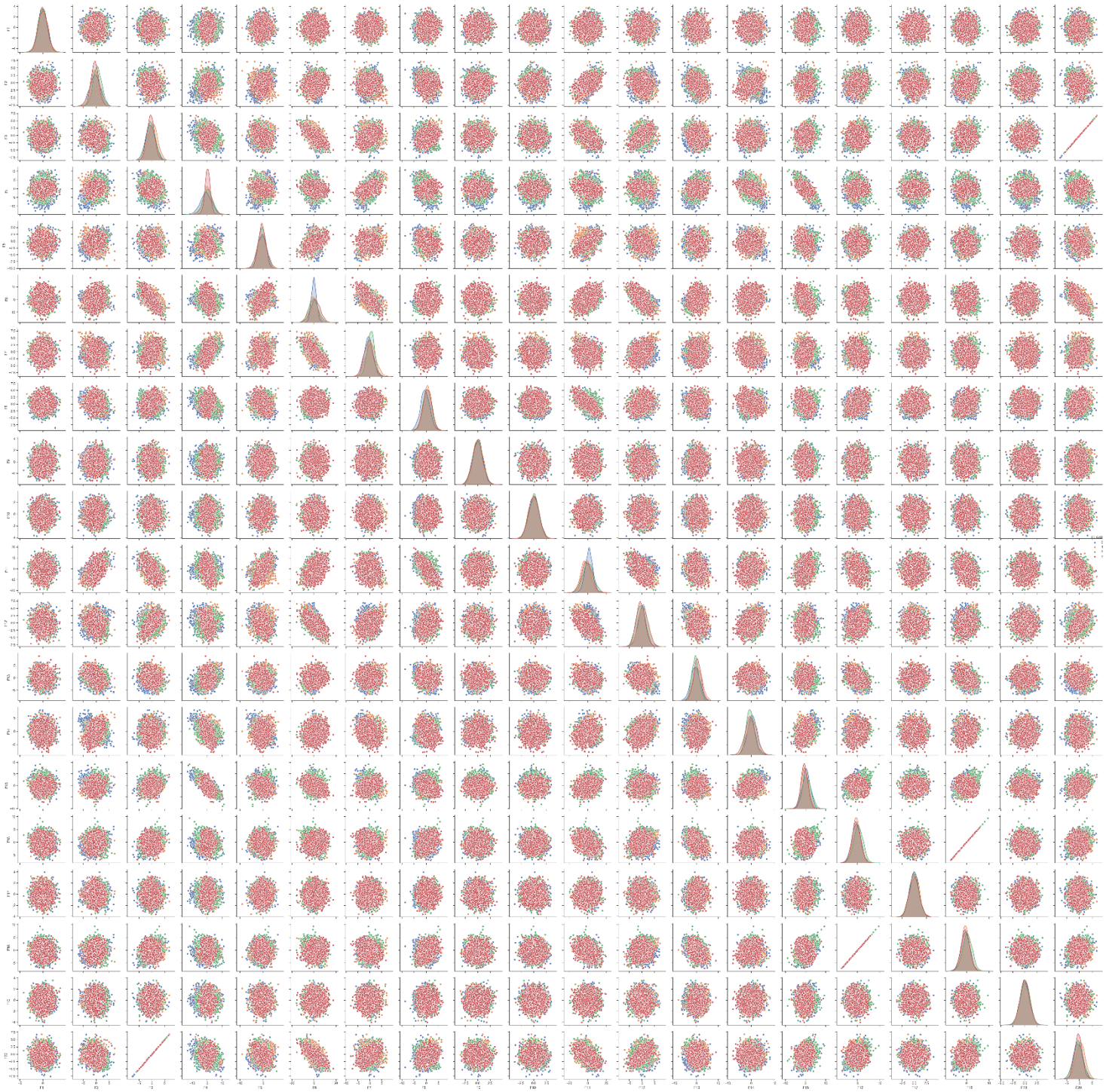
$$value = \frac{value - median}{p75 - p25}$$

In questo calcolo quindi non vengono considerati i valori assunti dagli outlier, che invece influenzano il calcolo della media e della deviazione standard nel caso dello StandardScaler.

Successivamente è stato utilizzato il MinMaxScaler con range (0,1), che ha mostrato risultati migliori del 5% su tutti i classificatori analizzati.

In seguito, si è passato alla feature selection, nella quale sono state scartate le feature che hanno una minor dipendenza con la feature target. Sono stati fatti dei test eliminando 10, 5 o nessuna feature, ottenendo il miglior risultato quando sono state eliminate 5 feature. Al termine di questa fase quindi ci saranno solo 15 feature: [F2,F3,F4,F5,F6,F7,F8,F11,F12,F13,F14,F15,F16,F18,F20].

La cosa può essere vista graficamente nel pairplot seguente, nel quale si vede che le feature eliminate – F1,F9,F10,F17,F19- sono quelle che hanno più overlap tra le varie classi e quindi non danno informazioni utili per la classificazione:



L'ultimo passaggio della fase di pre-processing è quello del balancing, ovvero del bilanciamento della numerosità delle classi target all'interno del training set. Questo processo viene eseguito per evitare che il modello finale sia poco addestrato nel riconoscere le classi meno presenti nel training set. In questo caso specifico, la numerosità delle classi era così distribuita:

- Classe 1: 33%
- Classe 2: 15%
- Classe 3: 20%
- Classe 4: 29%

Come tecnica di sampling è stato utilizzato SMOTE, in modo da non introdurre ridondanza nel dataset come nel caso dell'OverSampling e non privarsi di informazioni potenziamente rilevanti come nel caso dell'UnderSampling. Inoltre è stato testato anche l'utilizzo di Adasyn (che ha una logica simile a SMOTE), ma senza ottenere risultati rilevanti.

Quindi alla fine della fase di pre-processing il training set avrà 8572 record e 15 feature (invece dei 6400 record e 20 feature iniziali)

A questo punto sono stati scelti i modelli da utilizzare per la classificazione. La scelta è ricaduta su modelli visti a lezione oppure in altri corsi. In particolare, sono stati scelti:

- MultiLayerPerceptron
- Support Vector Machine
- K-nearest neighbrood
- Random Forest
- SGD (Stochastig Gradient Descent)

Per ognuno di essi, è stato fatto il tuning degli iper parametri utilizzando una 10-fold cross validation come mostrato in Figura 1:

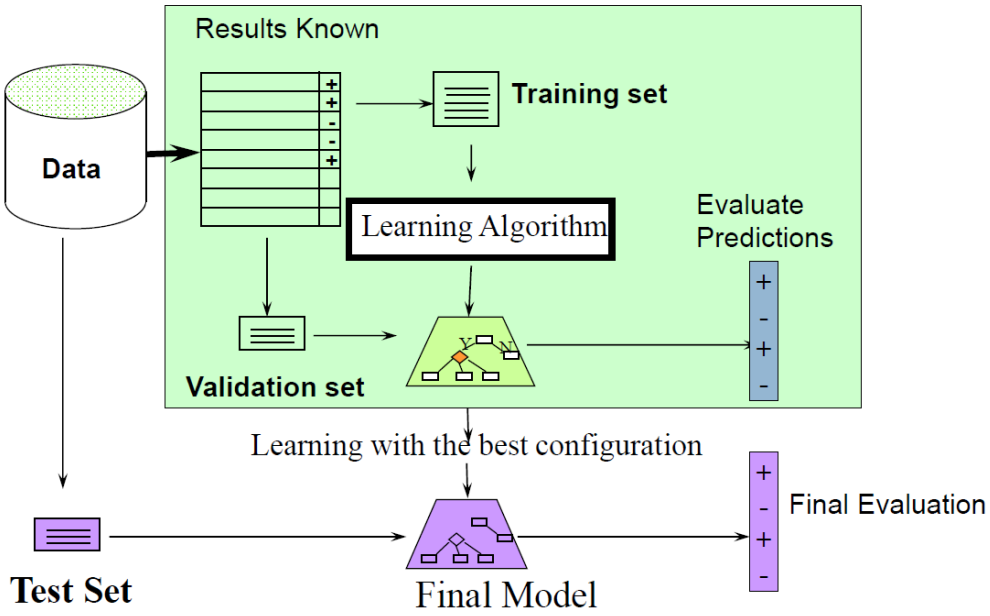


Figura 1. Valutazione del modello

Una volta separato il dataset iniziale in training set e test set e dopo aver processato i dati di training come descritto in precedenza, il training set viene ulteriormente suddiviso in training set e validation set (come illustrato nel rettangolo verde). A questo punto si addestra il modello con tutte le possibili combinazioni degli iperparametri desiderate, valutando il risultato con il validation set. Il modello verrà poi addestrato nuovamente con la combinazione degli iperparametri che ha dato uno score migliore della metrica F1 Macro e si valuteranno le sue prestazioni attraverso lo score ottenuto sul test set.

Alla fine, verrà scelto il modello che avrà dato il miglior risultato.

Per la scelta degli iperparametri da valutare, sono state fatte le seguenti considerazioni:

-MLP:

- ***Hidden layer size***: definisce il numero di layer ed il numero di percettroni presenti in ogni layer. Le architetture considerate sono state: [100,1] cioè 1 layer con 100 percettroni (architettura di deafult per sklearn), [100,50] (1 layer da 100 percettroni ed 1 layer da 50) e [100,50,25] (1 layer da 100 percettroni, uno da 50 ed uno da 25). Il motivo per cui ad ogni layer si è deciso di dimezzare il numero di percettroni è la complessità dal punto di vista computazionale
- ***Activation function:*** sono state scelte tanh e relu in quanto hanno una miglior resistenza al vanish gradient
- ***Solver:*** adam (default in sklearn) e sgd (stochastic gradient descent)

- ***Learning rate_init:*** misura quanto velocemente il modello si "adatta" ai dati di training. Più è piccolo, maggiore sarà la probabilità di ottenere un buono score, ma ci potrebbero volere molto epoche di training. Al contrario, più è grande e più velocemente si arriverà ad una soluzione, che però potrebbe non essere ottima. I learning rate scelti vanno da 0.1 a 0.0001
- ***Learning rate:*** si è scelto di utilizzare sia un approccio con learning rate costante (cioè utilizzando uno dei valori selezionati dal parametro Learning_rate_init) sia un approccio adattivo, cioè un approccio in cui si modifica il learning rate durante il processo di training in base alle prestazioni ottenute fino a quel momento

-SVM:

- ***Kernel:***definisce la funzione kernel con cui effettuare il kernel trick. Si è scelto di utilizzare il kernel lineare e quello rbf per testare gli effetti di un modello lineare rispetto ad uno non lineare.
- ***Decision function shape:*** per utilizzare una SVM in ambito di classificazione multiclasse, è necessario utilizzare un approccio OnevsRest o OnevsOne. Si è scelto di testare entrambi gli approcci.
- ***C:*** termine che stabilisce quanto penalizzare un errore di misclassificazione. Di conseguenza, più alto sarà questo valore e minore saranno gli errori permessi in fase di training. I valori utilizzati sono [0.1,1,10,25,50,75,100]
- ***Gamma:*** nel kernel rbf, definisce fino a che punto arriva l'"influenza" di un singolo esempio di training. Un valore alto di Gamma corrisponde al fatto che i punti più vicini all'iperpiano hanno maggior peso di rispetto a quelli più lontani. Viceversa, un valore basso di Gamma corrisponde al fatto che i punti più lontani hanno maggior peso rispetto a quelli più vicini. Si è scelto di utilizzare i valori da 10^{-4} a 10^{4}.

-RANDOM FOREST:

- ***Max_depth:*** profondità massima raggiungibile da un albero. Sono stati scelti valori da 10 a 150 con step di 10 per stimare come variano le prestazioni tra alberi di bassa profondità ed alberi con alta profondità.
- ***Max_features:*** massimo numero di features a cui guardare quando si sta determinando lo split migliore. Si è scelto di seguire gli approcci: auto (max_features = sqrt(n_features), log2 (max_features = log2(n_features)) e None (max_features = n_features)

- ***Min_sample_leaf:*** numero minimo di dati che possono essere assegnati ad una foglia. Si è scelto di utilizzare valori tra 4 e 12 con step di 2. In questo modo è possibile stimare la differenza tra un albero con pochi sample in una foglia rispetto ad uno con un buon numero di sample
- ***N_estimators:*** numero di alberi nella foresta. Si è scelto un valore tra 10 e 150 per stimare la differenza di prestazioni tra una foresta con pochi alberi ed una con molti alberi

-KNN:

- ***N_neighbors:*** numero di vicini da considerare. I valori considerati sono da 3 ad 11 con step di 2, in modo da considerare sia i casi in cui si considerano pochi vicini, sia i casi in cui se ne considerano molti
- ***Metrics:*** metriche per calcolare la distanza tra il punto da classificare ed i punti del training set. Si sono scelte alcune tra quelle disponibili su sklearn: minkowski, euclidea e chebyshev.
- ***P :*** parametro all'esponente della minkowski distance. Deve essere maggiore di 3, perché nel caso di p=2 la distanza di minkowsi diventa quella euclidea e nel caso p = 1 la distanza di minkowski diventa quella di manhattan. Per questo motivo si sono scelti i valori 3,4,5.

-SGD: Per questo classificatore, i parametri sono stati cercati in letteratura e settati come mostrato di seguito:

```
param_grid = {
    "loss": ["hinge", "log", "squared_hinge", "modified_huber"],
    'max_iter': [1000],
    'l1_ratio': [0.08, 0.09, 0.1, 0.12, 0.13, 0.14, 0.15, 0.2],
    "penalty": ["l2", "l1", 'elasticnet'],
}
```

Dato che per questo classificatore non ha dato buoni risultati in una prima fase di test, è stato scartato.

Risultati

Inizialmente lo score migliore è stato ottenuto dalla configurazione di Multi layer Perceptron mostrata in Figura 2, con dati scalati attraverso RobustScaler:

```
Best parameters:

{'activation': 'relu', 'hidden_layer_sizes': (100, 50), 'learning_rate': 'constant', 'learning_rate_init': 0.001, 'solver': 'sgd'}
```

Figura 2. Configurazione MLP

```
F1 for MLP: 0.8468563371177273
```

Figura 3. Risultato MLP

A seguire le prestazioni degli altri classificatori:

SVM:

```
Best parameters:

{'C': 10, 'decision_function_shape': 'ovo', 'gamma': 0.1, 'kernel': 'rbf'}
```

```
F1 for SVM : 0.8404261179424993
```

Figura 4. Risultato della SVM

RANDOM FOREST:

```
Best parameters:

{'criterion': 'entropy', 'max_depth': 60, 'max_features': 3, 'min_samples_leaf': 3, 'n_estimators': 300}
```

```
F1 for Random Forest 0.7845038848528102
```

Figura 5 Risultato di Random Forest

KNN:

```
Best parameters:

{'metric': 'minkowski', 'n_neighbors': 3, 'p': 3}
```

```
F1 for KNN 0.7845038848528102
```

Figura 6 Risultato KNN

Inoltre è stato testato anche il funzionamento di Adasyn (una variante di SMOTE) come procedura di oversampling per i classificatori Multi Layer Perceptron e SVM.

Di seguito i risultati:

MLP:

```
Best parameters:

{'activation': 'relu', 'hidden_layer_sizes': (100, 50, 25), 'learning_rate': 'adaptive', 'learning_rate_init': 0.1, 'solver': 'sgd'}
```

```
F1 for MLP: 0.8176642106992944
```

Figura 7 Risultati MLP

SVM:

```
Best parameters:

{'C': 100, 'decision_function_shape': 'ovo', 'gamma': 0.1, 'kernel': 'rbf'}
```

```
F1 for SVM : 0.7890948561972936
```

Figura 8 Risultati SVM

Come si può notare, l'utilizzo di Adasyn non ha portato miglioramenti.

Infine, è stato provato su MLP e SVM lo scaler MinMaxScaler con range (0,1), che ha dato i seguenti risultati:

-MLP

```
Best parameters:

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

```
F1 for MLP: 0.8672530412700951
```

Figura 9 Risultati MLP

-SVM

```
Best parameters:

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

```
F1 for SVM : 0.8085194557334857
```

Figura 10 Risultati SVM

Come si può notare, c'è stato un miglioramento del 5% su MLP, ma un peggioramento su SVM.

Dunque il miglior modello è un Multilayer Perceptron con parametri indicati nella figura 9, facendo uno scaling nel range (0,1).

Sviluppi futuri

-Migliorare la fase di feature selection utilizzando un approccio di tipo wrapper, il Recursive Feature Elimination o il SelectFromModel di sklearn.

-Valutare le prestazioni del classificatore utilizzando OverSampling o UnderSampling come tecniche di balancing

- Valutare le prestazioni del classificatore utilizzando altre tecniche di scaling

- Assegnare i punti mancanti nel dataset iniziale in base ad algoritmi di clustering come K-Means, invece di utilizzare la media dei valori assunti.

- Utilizzare altri classificatori come Adaboost o NaiveBayes

- Valutare il modello come in Figura1, ma utilizzando una cross validation anche per la valutazione finale (ovvero ripetere lo stesso processo descritto in Figura1, ma ripetuto cambiando il test set come in una cross validation e prendere come configurazione finale quella che risulta essere la migliore per il maggior numero di volte).