Multiclass Classification

Progetto MOBD 2019-2020

Marco Giorgi
Matricola: 0266724
Email: marco.giorgi90@outlook.com

In machine learning la classificazione multiclasse o multinomiale riguarda il problema di classificare le istanze in tre o più classi. Mentre molti algoritmi di classificazione consentono naturalmente l'uso di più di due classi, alcuni sono per natura algoritmi binari, ovvero nascono per classificare le istanze in sole due classi. Tuttavia questi algoritmi possono essere trasformati in classificatori multiclasse mediante una varietà di strategie, in particolare: One-vs-Rest (One-vs-All) e One-vs-One.

L'obiettivo del progetto è quello di risolvere un problema di classificazione multi-classe con 4 classi utilizzando algoritmi di machine learning.

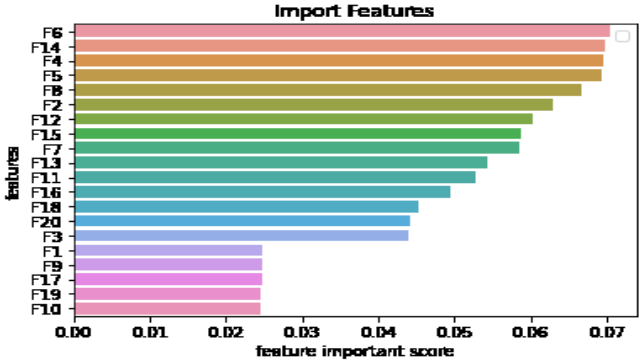

Fig. 1. Import Feature

1 Dataset

Il dataset proposto contiene 8000 tuple: vengono definite 20 feature che caratterizzano 4 classi numeriche da 0 a 3. I valori presentano una percentuale pari al 67.8% di esempi negativi e 32.2% di esempi positivi, sono presenti campi vuoti (tranne in quelli corrispondenti al campo "CLASS") e non sono presenti tuple duplicate. Gli elementi appartenenti alle classi sono così ripartiti :

classe 0: 2694
classe 1: 1279
classe 2: 1653
classe 3: 2374

Il dataset è stato generato artificialmente e completo presenta 10000 istanze.

Nella Fig 1 viene mostrata la percentuale di caratterizzazione delle feature sul target, in particolare le feature 'F10', 'F19', 'F17', 'F9', 'F1' hanno un impatto minore rispetto le altre. Mentre nella tabella successiva sono riportate per le Feature: la media, la deviazione standard e il numero di campi vuoti.

Feature	Mean	Std	NaN
F1	-0.013077	1.006235	6
F2	-0.261413	1.852793	6
F3	-0.356239	1.794600	1
F4	-0.107298	3.038362	1
F5	-0.505798	1.818965	4
F6	0.170845	3.802454	6
F7	-0.142636	1.901893	4
F8	0.135534	1.846124	9
F9	-0.004581	1.005507	6
F10	0.017338	1.005563	6
F11	-0.099617	5.072744	5
F12	0.000023	1.960223	6
F13	-0.051170	1.936503	3
F14	-0.011521	2.069112	10
F15	0.032990	2.013245	2
F16	-0.294768	1.970592	6
F17	-0.026807	0.978187	4
F18	-0.293875	1.971526	7
F19	0.001092	0.996603	5
F20	-0.355555	1.794666	3

2 Deploy Environment

Il progetto è stato svolto attraverso la piattaforma gratuita Google.Colaboratory utilizzando come linguaggio di programmazione Python. Principali librerie usate:

numpy: 1.18.5
pandas: 1.1.4
matplotlib: 3.2.2
scikit-learn: 0.22.2

3 Preprocessing

Una volta anlizzato il dataset, sono state apportate tutte le azioni necessarie ai fini dell'addestramento.

Dopo aver caricato e letto il file csv, sono stati individuati e sostituiti i valori NaN all'interno dei campi con la media dei valori presenti in ogni feature.

Sono stati quindi definiti due dataframe:

X : contenente gli elementi corrispondenti alle feature [F1...F20]

Y : contenente gli elementi corrispondeti al campo [CLASS].

Successivamente, i dataset sono stati suddivisi ognuno in due parti: la prima parte è stata utilizzata per il training del modello mentre la seconda per determinare gli iperparametri ottimali e valutare le prestazioni su una parte del dataset non vista nella fase di addestramento (la parte contenente il test non è presente). La proporzione 80%-20% scelta per lo split è stata conseguita dopo vari tentativi svolti al fine di migliorare i risultati. Per finire i dati sono stati opportunamente standardizzati.

4 Sperimentazione

Nella fase di sperimentazione sono stati individuati gli iperparametri necessari all'addestramento. In particolare, sono stati provati i classici algoritmi di multi-class e gli algoritmi binari utilizzando entrambi gli approcci one-vs-rest e one-vs-one.

La cross validation effettuata è del tipo k-cross validation con k pari a 10, valore individuato manualmente come miglior parametro in fase sperimentale.

Per il tuning dei parametri è stato pensato l'utilizzo della funzione GridSearchCV, ma per avere risultati più rapidi, per ogni algoritmo è stata implementata anche una funzione custom per riflettere più velocemente i risultati relativi alla f1.

La libreria sklearn propone diversi algoritmi per la classificazione multi-class, tra questi sono stati adottati:

DecisionTreeClassifier
LinearSVC
KNeighborsClassifier
SVM

I kernel delle SVM usati per il tuning dei parametri sono stati: poly (Polinomaile), rbf (Radial Basis Function), linear e sigmoid.

4.1 DecisionTreeClassifier

Gli alberi di decisione (DT) sono un metodo di apprendimento supervisionato non parametrico utilizzato per la classificazione e la regressione. Gli iperparametri necessari per la classe DT sono stati ottenuti utilizzando la GridSearchCV ed in particolare sono stati individuati:

max_depth
max_features
min_samples_leaf

Come descritto in [6] uno dei principali svantaggi dei DT è che possono essere instabili perché piccole variazioni nei dati potrebbero comportare la generazione di un albero completamente diverso. Questo problema viene mitigato utilizzando alberi di decisioni all'interno di un ensemble. Tra gli ensemble proposti sono stati utilizzati:

AdaBoostClssifier
BaggingClassifier
RandomForestClassifier

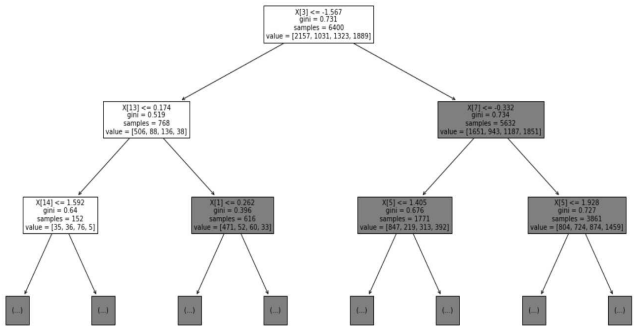

Fig. 2. Estratto del Tree ottenuto dal DecisionTreeClassifier

4.1.1 AdaBoostClassifier

Una volta ottenuti gli iperparametri opportuni per il DT questi sono stati applicati nella ricerca degli iperparemtri dell'AdaBoostClassifier, in particolare si è cercato di individuare il learning rate e gli n_estimators. Si è scelto di provare questo classificatore in quanto in generale il boosting migliora l'accuratezza della previsione su un problema multi-class.

L'AdaBoost Classifier messo a disposizione dalla libreria sklearn.ensemble adotta due algoritmi per la classificazione: SAMME e SAMME.R.

Come descritto in [7], l'algoritmo SAMME.R tipicamente converge più velocemente rispetto a SAMME, ottenendo un errore di test inferiore con meno iterazioni di boost.

4.1.2 BaggingClassifier

Poiché forniscono un modo per ridurre l'overfitting, i metodi di Bagging funzionano meglio con modelli forti e

complessi (ad esempio, alberi decisionali completamente sviluppati), in contrasto con i metodi di boost che di solito funzionano meglio con modelli deboli (ad esempio gli shallow decision trees). Sono stati individuati per questo classificatore: il numero di stimatori di base nell'ensamble e la profondità massima.

4.1.3 RandomForestClassifier

Per quanto riguarda i RandomForest la ricerca degli iperparametri è stata fatta variando i valori relativi alla profondità dell'albero ed il numero di alberi all'interno della "Forest".

4.2 KNeighborsClassifier

La KNeighborsClassifier (KNN) è un classificatore che implementa la k-nearest neighbors vote e fa parte della libreria sklearn.neighbors che fornisce funzionalità per metodi di apprendimento basati sui neighbors in modo supervisionato e non supervisionato.

La KNeighborsClassifier (KNN), come descritto in [2], è un tipo di apprendimento basato sull'istanza o apprendimento non generalizzante, la classificazione viene calcolata da un voto a maggioranza semplice dei neighbors più vicini di ogni punto: ad un punto viene assegnata la classe di dati che ha il maggior numero di rappresentanti all'interno dei neighbors più vicini del punto.

Gli algoritmi usati per calcolare i nearest neighbors sono di tre tipi: il BallTree, il KDTree e la ricerca a forza bruta. Nel caso in esame è stato utilizzato il parametro "auto" in modo tale da decidere l'algoritmo più appropriato in base ai valori passati al metodo fit.

La scelta di k dipende dalle caratteristiche dei dati. Generalmente all'aumentare di k si riduce il rumore che compromette la classificazione, tuttavia il criterio di scelta per la classe diventa più labile. La scelta del numero di neighbors e quindi di k è presa nel nostro caso attraverso la k-cross validation.

Si è scelto di combinare insieme al KNN anche la Neighborhood Components Analysis (NCA, NeighborhoodComponentsAnalysis), un algoritmo di apprendimento metrico della distanza che mira a migliorare l'accuratezza della classificazione dei neighbors più vicini rispetto alla distanza euclidea standard.

NCA viene descritto da [3] come un metodo di apprendimento supervisionato per classificare i dati multivariati in classi distinte in base ad una data metrica di distanza sui dati. Funzionalmente, ha gli stessi scopi dell'algoritmo dei K-nearest neighbors e fa uso diretto della stochastic nearest neighbours.

Sebbene questa classe di algoritmi di machine learning non è stata trattata nel corso è stata scelta di utilizzarla in quanto molto consigliata per i problemi di multi-class e perchè semplice nel suo funzionamento.

4.3 SVM e LinearSVC

LinearSVC è simile a SVC con parametro kernel = 'linear', ma è implementato in termini di liblinear piuttosto che libsvm, quindi ha una maggiore flessibilità nella scelta delle penalità e delle funzioni di perdita e dovrebbe scalare meglio a un gran numero di campioni.

Questa classe supporta il multi-class con uno schema "ovr" di default e con uno schema "crammer_singer" come descritto in [4].

Per quanto riguarda le prestazione dell'SVM con kernel rbf, la scelta corretta di C e di gamma è fondamentale, mentre per le SVM con kernel polinomiale i parametri fondamentali da individuare sono il degree e il coefficiente C. Per il kernel linear e il kernel sigmoid è stato cercato in modo opportuno solo il parametro C.

Un valore basso di C rende smooth la superficie decisionale, mentre un alto C mira a classificare correttamente tutti gli esempi di training.

Il valore gamma definisce quanta influenza ha un singolo esempio di train.

4.3.1 One-vs-Rest & One-vs-One

La One-vs-Rest conosciuta anche come la One-vs-All implica la suddivisione del set di dati multi-class in più problemi di classificazione binaria. Un classificatore binario viene quindi addestrato su ogni problema di classificazione binaria e le previsioni vengono effettuate utilizzando il modello più affidabile.

Questo approccio richiede che ogni modello preveda una probabilità di appartenenza a una classe o un punteggio simile alla probabilità. L'argmax di questi punteggi viene quindi utilizzato per prevedere una classe.

Un possibile svantaggio di questo approccio è che richiede la creazione di un modello per ogni classe. Ad esempio, tre classi richiedono tre modelli. Questo potrebbe essere un problema per set di dati di grandi dimensioni (ad esempio milioni di righe), modelli lenti (ad esempio reti neurali) o un numero molto elevato di classi (ad esempio centinaia di classi). Dato che il dataset trattato in questo progetto non presenta grandi quantità di dati è stata presa in considerazione anche la strategia One-vs-Rest (ovr).

La libreria scikit-learn fornisce anche una classe OneVsRestClassifier separata che consente di utilizzare la strategia One-vs-Rest con qualsiasi classificatore.

Il One-vs-One (ovo) è un altro metodo euristico per l'utilizzo di algoritmi di classificazione binaria per la classificazione multi-class. Questa strategia divide un set di dati in problemi di classificazione binaria: un set di dati per ogni classe rispetto a ogni altra classe, a differenza del One-vs-Rest.

Solitamente i SVC e NuSVC implementano l'approccio "uno contro uno" per la classificazione multi-class. Anche in questo caso la libreria scikit-learn fornisce una classe OneVsOneClassifier separata che consente di utilizzare la strategia "uno contro uno" con qualsiasi classificatore.

5 Risultati

I risultati sono stati ottenuti calcolando opportunamente la f1-macro. Per ogni algoritmo nella seguente tabella viene riportato il valore migliore ottenuto nel corso della sperimentazione, i dati fanno riferimenti solo ai risultati conseguti dalla cross validation in quanto non è presente il completo dataset.

Algoritmo	Specifiche	F1	Accuratezza
KNN	NCA	0.8268	0.845
RandomForest		0.799	0.81625
AdaBoost	SAMME	0.814	0.829
AdaBoost	SAMME_R	0.787	0.805
Bagging		0.784	0.80
SVC_Rbf	ovr	0.839	0.8525
SVC_Rbf	ovo	0.825	0.8412
SVC_Poly	ovo	0.8415	0.856
SVC_Poly	ovo	0.826	0.845
SVC_linear	ovr	0.425	0.5
SVC_linear	ovo	0.527	0.5768
SVC_sigmoid	ovr	0.359	0.415
SVC_sigmoid	ovo	0.351	0.416
LinearSVC	ovr	0.485	0.53

6 Note

La Fig 1 che rappresenta l'importanza delle feature è stata generata utilizzando la proprietà feature_importances della classe di algoritmi RandomForestClassifier: l'importanza della feature, nota come Gini importance, è basata sull'impurità; più alta è l'impurità, più importante è la feature (vedere [5]).

7 Conclusioni

Nelle ricerche condotte e in particolare in [1] si è osservato che i problemi di tipo multi-class possono essere risolti con molti approcci, ad esempio con reti neurali o con algoritmi naive bayes o con classificazioni gerarchiche ecc... Oltre agli algoritmi utilizzati sarebbe stato interessante anche osservare i risultati prodotti da tutti gli altri approcci.

Le difficoltà riscontrate sono per lo più dovute ai limiti della piattaforma utilizzata, oltre che alla mancanza di uno standard vero e proprio per la trattazione del problema multi-class.

A livello di ricerca è stato utile osservare, oltre gli esempi proposti nel corso, anche quelli relativi al dataset iris a cui molti progetti Multi-Class-Classification fanno riferimento.

References

[1] Multiclass and multilabel algorithms
https://scikit-learn.org/stable/modules/multiclass.html

[2] Nearest Neighbors
https://scikit-learn.org/stable/modules/neighbors.html#classification

[3] Neighbourhood components analysis
https://en.wikipedia.org/wiki/Neighbourhood_components_analysis

[4] LinearSVC
https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
https://scikit-learn.org/stable/modules/svm.html#svm-classification

[5] RandomForsetClassifier
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
https://www.datacamp.com/community/tutorials/random-forests-classifier-python

[6] DecisionTreeClassifier
https://scikit-learn.org/stable/modules/tree.html#tree
https://scikit-learn.org/stable/modules/ensemble.html#forest

[7] Ji Zhu , Hui Zou, Saharon Rosset and Trevor Hastie *"Multi-class AdaBoost"*. 2009