“Analisi di differenti modelli”

Relazione progetto ALW

Cristian Milia, Emanuele Camarda

Indice

Introduzione

Il Machine Learning è un insieme di metodi per migliorare progressivamente la performance di un algoritmo nell'identificare pattern nei dati. Il Machine Learning è composto da molte branche,due delle quali sono la classificazione e la regressione, che a partire da alcuni esempi, che formano un dataset, cercano di capire la classe o il valore delle caratteristiche di nuovi esempi che i modelli di classificazione e regressione non hanno mai visto.

Per la classificazione o la regressione si usano dei differenti modelli di Machine Learning che seguendo degli algoritmi dipendenti dal modello scelto tentano di classificare o fare regressione sui nuovi esempi.

La bontà di un modello rispetto ad un altro è valutabile con differenti metriche e dipende da numerosi fattori come la grandezza del dataset, la presenza di valori mancanti, il numero delle classi considerate e così via, per questo, in generale, non esiste un modello migliore in tutti i casi rispetto agli altri modelli.

Inoltre, c'è da considerare che ogni metrica misura proprietà diverse di un modello; può accadere che un modello dia risultati ottimi rispetto una metrica ma non rispetto un'altra. Quindi la scelta di un modello "migliore" non è scontata, in quanto non è possibile definire un metodo univoco di valutazione di un modello.

Lo scopo di questo progetto è analizzare differenti modelli di Machine Learning su alcuni dataset di classificazione e regressione, al fine di capire, considerando differenti metriche, quali modelli sono più robusti nel caso di dati mancanti, e quali si comportano bene nella pratica su diversi tipi di dataset.

Per prima cosa verranno illustrate le caratteristiche dei dataset analizzati e il modo in cui essi vengono importati e modificati. Successivamente verranno presentate tutte le metriche di classificazione e regressione e i relativi modelli di Machine Learning studiati.

A quel punto si mostrerà come viene scelto il modello migliore di Machine Learning in funzione del valore di un insieme di metriche, e quali problemi comporta tale scelta.

Infine, verranno discussi e presentati i risultati del progetto in forma grafica per mezzo di radarplot e tabelle.

Tecnologie di sviluppo

Per la realizzazione del progetto di Machine Learning è stato utilizzato esclusivamente il linguaggio di programmazione Python tramite l'ide PyCharm. Le librerie principali utilizzate sono:

- Scikit-learn, una libreria di machine learning che ha permesso l'uso dei modelli di classificazione e regressione con le relative metriche;
- Matplotlib, una libreria per la creazione dei radarplot;
- Plotly e imgkit, lbrerie per la creazione delle tabelle;
- Csv, libreria per la lettura dei file csv.

Dataset

Descrizione Dataset

La prima cosa che un modello di Machine Learning ha bisogno è il dataset.

Sono stati analizzati 6 dataset per la tecnica della classificazione e 5 per la tecnica della regressione, diversificati per numero di caratteristiche e per il numero di esempi che ogni dataset contiene.

I dataset sono stati inizialmente scaricati in formato CSV.

Tra i dataset di classificazione troviamo:

- Balance con 625 esempi e 4 features
- Eye con 14979 esempi e 15 features
- Page con 5473 esempi e 10 features
- Seed con 210 esempi e 7 features
- Tris con 957 esempi e 9 features
- Zoo con 100 esempi e 18 features

Invece, tra i dataset di regressione troviamo:

- Airfoil con 1502 esempi e 5 features
- Auto con 398 esempi e 9 features
- Power Plant con 9568 esempi e 4 features
- Compressive Strength con 1030 esempi e 8 features
- Energy con 768 esempi e 8 features

Nessun dataset presenta valori mancanti.

Caricamento Dataset

I dataset nei file csv vengono importati utilizzando una funzione read_csv() che oltre alla possibilità della scelta del delimitatore riesce a gestire dataset multilabel, non considerare delle colonne a sinistra o a destra e saltare un certo numero di righe di intestazione.

Nel caso in cui il dataset non contenga esclusivamente numeri, non è possibile darlo in input al modello, per cui viene passata una lista di valori di rimpiazzo, dove per ogni occorrenza di un valore non numerico nel dataset originale, esso viene rimpiazzato con uno dei valori numerici preso dalla lista di rimpiazzo, in modo tale che valori non numerici uguali vengano rimpiazzati con lo stesso valore. Tale operazione è stata eseguita ad esempio nel dataset “tris” dove erano presenti i valori ‘x’, ‘o’ e ‘b’.

Una volta importato il dataset, viene applicata una modifica al dataset e viene dato in input ai modelli che calcolano le metriche di classificazione e/o regressione.

Le modifiche al dataset sono fondamentali per capire come varia il modello migliore non solo al variare del tipo di dataset, ma anche al variare della modifica apportata ad esso. Ad esempio un dataset può essere classificato molto bene con RandomForest, ma non appena vengono messi a

NaN(Not A Number) il 5% dei suoi valori, può risultare che viene classificato meglio con SVM(Support Vector Machine).

One Hot Encoding

Una prima modifica che viene apportata ai dataset di classificazione multiclasse è il ”one hot encoding”.

Esso consiste nel trasformare un dataset multiclasse in uno multilabel in cui ogni label ha 0 o 1 come possibili valori, tale tecnica, scorrelando i valori delle labels tra loro, permette di non far confondere il modello, migliorando così la sua predizione.

Il modello, senza applicare one hot encoding, può confondersi nel senso che può trovare una relazione tra i valori delle label e i valori delle features, che in realtà non esiste. Ad esempio un dataset con possibili valori delle label “1,2,4” potrebbe sfruttare il fatto che 1 è la metà di 2 e 2 è la metà di 4, quando in realtà 1 era stato associato alla classe “cane”, 2 alla classe “gatto” e 4 alla classe “persona”, in cui non ha minimamente senso una relazione matematica tra le varie classi.

Dataset completi

Le modifiche al dataset si differenziano in base a come si vuole considerare il dataset: se si vuole mantenere il dataset con tutti gli esempi oppure se lo si vuole analizzare con una percentuale di dati mancanti. Nel caso dei dataset mancanti, siccome nessun dataset presenta valori NaN per studiarne uno è sufficiente rimpiazzare i valori di alcune caratteristiche con valori NaN.

Per i dataset completi le modifiche sono principalmente due: la normalizzazione e la standardizzazione.

La normalizzazione modifica il dataset in modo tale che i valori di una caratteristica siano compresi nell’intervallo [0,1] estremi inclusi, dove 0 è associato al valore della caratteristica minore e 1 al valore della caratteristica maggiore.

$$normalized_{value} = \frac{value - \min}{max - min}$$

$se\ value = \max\ \ normalized_{value} = 1$

$se\ value = \min\ \ normalized_{value} = 0$

La standardizzazione invece applica la formula:

$$standardized_{value} = \frac{value - mean}{stdev}$$

$$mean = \frac{\sum value_i}{num_{values}}$$

$$stdev = \sqrt{\sum \frac{(value_i - mean)^2}{(num_{values} - 1)}}$$

La standardizzazione scala i valori in modo che la media di questi sia 0 e la deviazione standard sia 1.

Queste 2 modifiche sono importanti, perché è possibile che alcuni dataset vengono classificati in modo migliore/peggiore se viene applicata la normalizzazione o la standardizzazione.

Dataset con dati mancanti

I dataset mancanti, come detto precedentemente, si ottengono ponendo alcuni valori delle statistiche a NaN. In particolare, sono state scelte 3 percentuali (5%,10%,15%) che rappresentano la percentuale di valori NaN che dovranno essere presenti nel dataset con valori mancanti.

Dopo aver scelto le percentuali, in maniera del tutto casuale vengono scelti dei valori delle caratteristiche del dataset da porre a NaN.

Tuttavia, i modelli non funzionano con i valori NaN per cui occorre applicare delle strategie per sostituire il valore NaN con un valore ammissibile, una volta fatto ciò è possibile eseguire il modello su quel dataset.

Le strategie adottate sono 4:

- "Eliminate_row" che elimina l'esempio del dataset che contiene almeno un valore NaN;
- "Mode" che rimpiazza il valore NaN con il valore più frequente presente nella colonna dove è presente il valore NaN, ossia seleziona il valore più frequente tra tutti i valori di una caratteristica;
- "Median" che rimpiazza il valore NaN con la mediana dei valori nella colonna dove è presente il valore NaN;
- "Mean" che rimpiazza il valore NaN con la media dei valori nella colonna dove è presente il valore NaN;

Da notare che non su tutti i dataset è possibile applicare tutte le strategie, per esempio sul dataset "zoo" è possibile applicare solo le strategie "eliminate_row" e "Mode", in quanto i valori di tutte le caratteristiche variano su un insieme discreto. Inoltre, per alcuni dataset, vengono applicate le strategie "mean" o "median" congiuntamente alla strategia "mode", nel caso in cui su una colonna specifica non ha senso applicare "mean" o "median", perché i valori variano su un insieme discreto.

Classificazione

In questo capitolo andremo a trattare il caso del problema di classificazione, andando ad analizzare i modelli scelti, le metriche utilizzate e le fasi di training, testing e analisi dei risultati ottenuti.

Modelli analizzati

Nella realizzazione del nostro progetto siamo andati ad analizzare e testare diversi modelli di machine learning, per poterli poi confrontare in termini di prestazioni sui diversi dataset scelti. In particolare, siamo andati ad utilizzare dalla libreria *scikit learn* le seguenti implementazioni:

- sklearn.tree.DecisionTreeClassifier: implementazione di un albero decisionale con uso del criterio di "*Gini impurity*" per misurare la qualità di uno split;
- sklearn.neighbors.KNeighborsClassifier: implementazione del modello KNN (K-Nearest Neighbors) con input il valore assegnato a k e uso della distanza tra due punti per la scelta del vicinato;
- sklearn.esemble.RandomForestClassifier: implementazione del modello random forest con input il numero di alberi della foresta (trees) e il numero di features da usare nella costruzione di un singolo albero (max_features);
- sklearn.svm.SVC: implementazione di una Support Vector Machine, con parametri di input il tipo di kernel (kernel) e i parametri di setting della particolare funzione scelta:
    - kernel=*"linear"*: parametro di penalty *"C"* per la scelta del termine di errore;
    - kernel=*"poly"*: parametro di penalty *"C"* per la scelta del termine di errore, il grado *"degree"* del polinomio di approssimazione, il parametro *"gamma"* e il parametro *"coef0"*;
    - kernel=*"rbf"*: parametro di penalty *"C"* per la scelta del termine di errore e il parametro *"gamma"*;
    - kernel=*"sigmoid"*: parametro di penalty *"C"* per la scelta del termine di errore, il parametro *"gamma"* e il parametro *"coef0"*.

Metriche analizzate

Per la valutazione della bontà di un modello, abbiamo utilizzato diverse metriche, ognuna delle quali va a valutare una determinata qualità della predizione.

Di seguito sono riportate le metriche utilizzate, ove nelle formule con TP = Veri Positivi, FP = Falsi Positivi, TN = Veri Negativi e FN = Falsi Negativi:

- Accuracy: misura l'accuratezza di una predizione, ovvero le predizioni corrette su il numero totale di dati.

$$Accuracy = \frac{TP + FN}{TP + FP + TN + FN}$$

- Precision: misura il numero di elementi predetti bene di una classe su il numero totale di elementi di quella classe.

$$Precision = \frac{TP}{TP + FP}$$

- Recall: misura il numero di elementi predetti bene di una classe su il numero totale di elementi predetti come appartenenti a quella classe dal classificatore.

$$Recall = \frac{TP}{TP + FN}$$

- Average_precision: misura l'area sotto la curva precision-recall. In formule abbiamo:

$$\sum_{n} (R_n - R_{n-1}) P_n$$

  Ove R_n è il valore di recall, mentre P_n è il valore della precision.

- F1: misura la media armonica dei valori di precision e recall. In formule abbiamo:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

- ROC AUC: misura l'area sotto la curva del grafico che ha per ascisse i FP, e per ordinate i TP, ovvero misura il rapporto tra gli elementi predetti correttamente e quelli predetti erroneamente come falsi.

Osservare che tutte queste metriche restituiscono un valore compreso tra 0 e 1, e sono tutte del tipo *"greater is better"*. Inoltre, per il caso di problema multiclasse, è stata utilizzata una strategia *"micro"* per unire i risultati, ossia si fa una media globale, ovvero pesata sul numero di elementi di una classe, dei vari valori della metrica ottenuti.

Training

Nella fase di training si vanno ad allenare, uno per uno, i vari modelli scelti su, ognuno dei dataset precedentemente descritti. Inoltre, al fine di ottenere il miglior setting possibile per una determinata coppia modello-dataset, si sono effettuati diversi training facendo variare i parametri di input, andando poi a selezionare il modello migliore.

La scelta del modello migliore avviene tramite il calcolo di un *"total_score"*, ovvero di un numero che in sé racchiuda la bontà delle varie metriche. Il setting di una coppia modello-dataset con total_score maggiore verrà scelto come migliore. Come total_score abbiamo scelto di utilizzare la media armonica delle sei metriche scelte, in quanto tutte comprese tra 0 e 1 e in modo da poter scegliere il setting che "globalmente" va meglio su ogni metrica scelta.

Ricordiamo che la media armonica H dei numeri reali positivi $x1, x2, \dots, xn$ è definita come:

$$H = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} = \frac{n \cdot \prod_{j=1}^{n} x_j}{\sum_{i=1}^{n} \frac{\prod_{j=1}^{n} x_j}{x_i}}.$$

Quindi nel nostro caso abbiamo:

$$H = \frac{6}{\left(\frac{1}{Accuracy} + \frac{1}{Precision} + \frac{1}{Recall} + \frac{1}{AvgPrecision} + \frac{1}{F1} + \frac{1}{RocAuc}\right)}$$

Andiamo ad analizzare come sono stati fatti variare i parametri di input dei vari modelli nella fase di training:

- CART: non ci sono parametri di input significativi che sono stati fatti variare;
- Random Forest: i parametri di input sono il numero di alberi nella foresta, il quale varia in un range compreso tra 5 e 20, e il numero di features utilizzate per la creazione di ogni albero, che varia in un range compreso tra 1 e il numero totale di features;
- KNN: l'unico parametro di input è il numero k dei vicini, fatto variare in un range compreso tra 3 e 20;
- SVC: i vari parametri inseriti, utilizzati in funzione del nucleo scelto, sono:
    - Kernel, scelto tra "linear", "poly", "rbf", "sigmoid";
    - C, scelto in un range logaritmo compreso tra 10^{-2} e 10^{2};
    - Gamma, scelto in un range logaritmico compreso tra 10^{-5} e 1;
    - Degree, scelto tra 2 e 3;
    - Coef0, scelto costantemente uguale a 0.

La scelta di tali parametri è stata presa sulla base di documentazione di altri test di training trovati online e della teoria studiata in merito.

Da un punto di vista implementativo, la fase di training viene svolta all'interno del main del file *"training.py"*, nel quale avvengono due fasi principali:

1. Import e modifica del dataset scelto dal file csv, tramite i metodi *"resultAnalysis.case_full_dataset"* per il caso di dataset completi con eventuali strategie di normalizzazione o standardizzazione, e *"resultAnalysis.case_NaN_dataset"* per il caso di dataset con dati mancanti (5%, 10% o 15% di NaN), con strategia di filling scelta tra "mode", "mean", "median" e "eliminate_row".
2. Training di ogni modello per il particolare dataset scelto, tramite il metodo *"training.training"*, in cui ogni modello viene ciclato su tutti i possibili input. I setting migliori, scelti tramite il total_score, vengono salvati in dei file di setting, contenuti nella cartella *"best_model_settings/classification_model_settings"*, insieme ai valori delle metriche ottenuti.

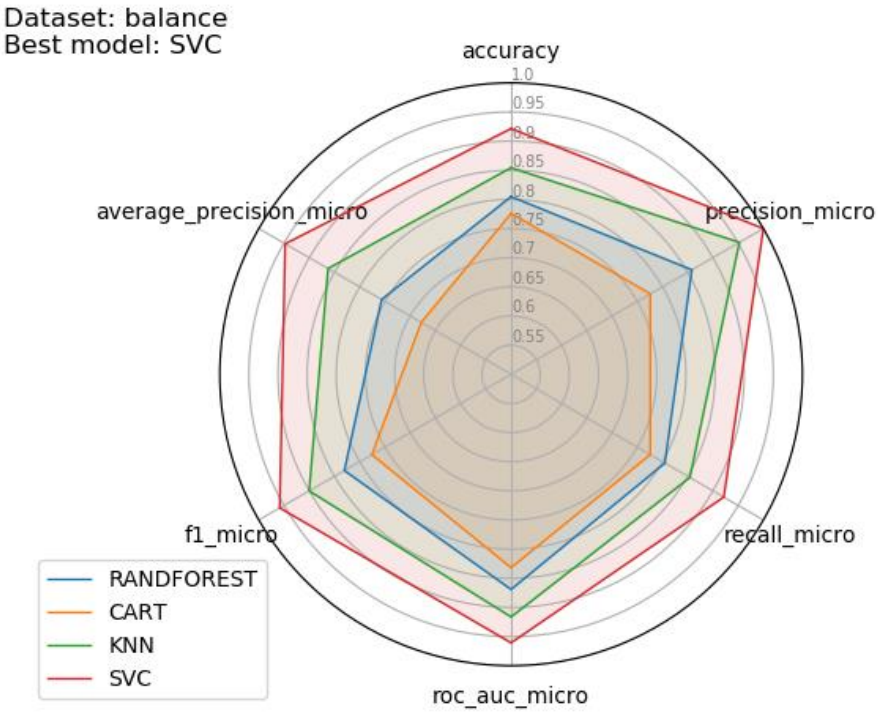
Osserviamo che, per poter usare SVC, è stato necessario applicarci il metodo *"sklearn.multiclass.OneVsRestClassifier"* per poterlo convertire, tramite la strategia One Vs Rest, da un classificatore multiclass a uno multilabel.

Il training unitario di un singolo setting per un modello viene effettuato utilizzando la tecnica del K-Fold-Cross-Validation. Ciò è stato realizzato grazie al supporto offerto dalla libreria scikit-learn, in particolare grazie alle funzioni *"sklearn.model_selection.KFold"* per creare i fold (nel nostro caso 10), e *"sklearn.model_selection.cross_validate"*. La tecnica del K-Fold-Cross-Validition è utile per diminuire l'overfitting del modello scelto, facendo variare quelli che sono i dati usati per il training e quelli per il testing, e utilizzando poi la media dei risultati ottenuti.

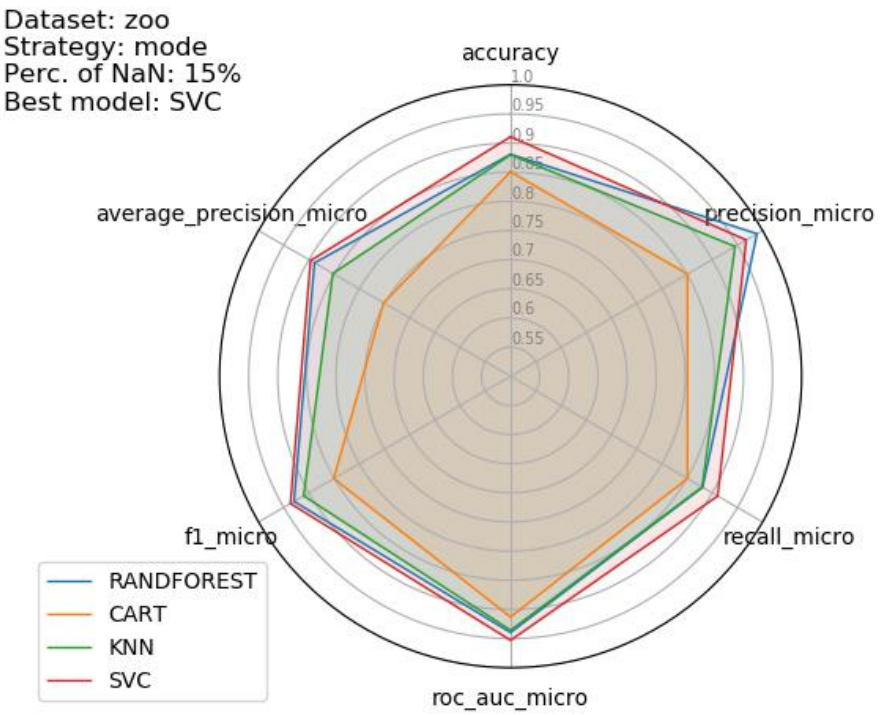
Risultati ottenuti

La fase di analisi dei risultati viene effettuata all'interno del main del file *"resultAnalysis.py"*. In particolare, vengono letti tutti i vari file di setting per poter creare dei radar plot riassuntivi dei risultati ottenuti. Ogni radar plot contiene le varie metriche usate come score, e in colori diversi contiene i risultati ottenuti dal setting migliore di ogni modello. Inoltre, sono presenti nella legenda in alto a sinistra il nome del dataset, il modello migliore, la strategia adottata ed eventualmente la percentuale di NaN. Infine, viene realizzata una tabella finale riassuntiva contenente il miglior modello, scelto sempre sulla base del total_score, per ogni dataset, e modifica di quest'ultimo, analizzato.

Ecco alcuni risultati ottenuti:



In questo radar plot si può notare che i valori delle metriche relative al modello SVC sono maggiori rispetto a qualsiasi altro modello.



Invece, in quest'altro esempio, nonostante SVC abbia total_score maggiore, ha un valore di precision più basso di KNN e Random Forest.

Questa invece è la tabella riassuntiva per quanto riguarda la classificazione:

Summary table classification

classification	seed	tris	zoo	balance	eye	page
full	SVC	SVC	RANDFOREST	SVC	KNN	RANDFOREST
normalize	SVC	SVC	RANDFOREST	SVC	RANDFOREST	RANDFOREST
standardize	SVC	SVC	RANDFOREST	SVC	RANDFOREST	RANDFOREST
5.0%_eliminate_row		SVC		SVC	KNN	RANDFOREST
5.0%_mean	SVC			SVC	KNN	RANDFOREST
5.0%_median	SVC			SVC	KNN	RANDFOREST
5.0%_mode		RANDFOREST	SVC	SVC		
10.0%_eliminate_row		RANDFOREST		SVC	SVC	RANDFOREST
10.0%_mean	RANDFOREST			SVC	KNN	RANDFOREST
10.0%_median	RANDFOREST			SVC	KNN	RANDFOREST
10.0%_mode		RANDFOREST	SVC	SVC		
15.0%_eliminate_row		RANDFOREST		SVC	SVC	RANDFOREST
15.0%_mean	RANDFOREST			SVC	KNN	RANDFOREST
15.0%_median	SVC			SVC	KNN	RANDFOREST
15.0%_mode		RANDFOREST	SVC	SVC		

Analizzando la tabella, possiamo notare che su dataset diversi, sono migliori modelli diversi; ad esempio per il dataset “balance” è migliore il modello SVC per ogni strategia applicata, mentre per il dataset “page” è sempre migliore il modello Random Forest.

Inoltre, per alcuni dataset, si può cogliere che la presenza o meno di dati mancanti influisce sulla determinazione del modello migliore; ad esempio in “tris”, senza eliminare dati il modello migliore è SVC, altrimenti Random Forest; analogamente per il dataset “zoo” abbiamo la stessa situazione precedente, ma con ruoli invertiti.

Regressione

Similarmente al capitolo precedente, andremo a trattare il caso del problema di regressione, andando ad analizzare i modelli scelti, le metriche utilizzate e le fasi di training, testing e analisi dei risultati ottenuti.

Modelli analizzati

I modelli scelti per il caso della regressione sono analoghi a quelli della classificazione. In particolare, siamo andati ad utilizzare dalla libreria *scikit learn* le seguenti implementazioni:

- sklearn.tree.DecisionTreeRegressor: implementazione di un albero decisionale per un problema della regressione;
- sklearn.neighbors.KNeighborsRegressor: implementazione del modello KNR (K-Nearest Neighbors-Regressor) per un problema di regressione, con input il valore assegnato a k e uso della distanza tra due punti per la scelta del vicinato;
- sklearn.esemble.RandomForestRegressor: implementazione del modello random forest per un problema di regressione, con input il numero di alberi della foresta (trees) e il numero di features da usare nella costruzione di un singolo albero (max_features);
- sklearn.svm.SVR: implementazione di una Support Vector Machine per un problema di regressione, con parametri di input il tipo di kernel (kernel) e i parametri di setting della particolare funzione scelta:
    - kernel=*"linear"*: parametro di penalty *"C"* per la scelta del termine di errore;
    - kernel=*"poly"*: parametro di penalty *"C"* per la scelta del termine di errore, il grado *"degree"* del polinomio di approssimazione, il parametro *"gamma"* e il parametro *"coef0"*;
    - kernel=*"rbf"*: parametro di penalty *"C"* per la scelta del termine di errore e il parametro *"gamma"*;
    - kernel=*"sigmoid"*: parametro di penalty *"C"* per la scelta del termine di errore, il parametro *"gamma"* e il parametro *"coef0"*.

Metriche analizzate

Per la valutazione della bontà di un modello, siamo andati ad usare diverse metriche, ognuna delle quali va a valutare una determinata qualità della predizione. Di seguito sono riportate le metriche utilizzate:

- Explained Variance (EA): misura come la varianza del modello predetto si discosti dalla varianza del modello reale, ed è molto correlata alla metrica R2 (uguali se l'errore medio delle predizioni è 0). Assume valori da -inf a 1:

$$EA = 1 - \frac{{}^{1}/_{n}\sum_{n}[(y_n - \hat{y}_n)^2 - mean(y - \hat{y})]}{{}^{1}/_{n}\sum_{n}(y_n - \bar{y}_n)^2}$$

- Negative Mean Absolute Error (-MAE): misura la media del valore assoluto della differenza tra i valori predetti e quelli reali. Assume valore tra -inf e 0:

$$-MAE = -\frac{1}{n}\sum_{n}|y_n - \hat{y}_n|$$

- Negative Mean Squared Error (-MSE): misura la media della differenza al quadrato tra i valori predetti e quelli reali. Assume valore tra -inf e 0:

$$-MSE = -\frac{1}{n}\sum_{n}(y_n - \hat{y}_n)^2$$

- R2: misura come la varianza del modello predetto si discosti dalla varianza del modello reale. In altre parole, esprime quanto bene le variabili indipendenti del modello esprimano la varianza delle variabili dipendenti. Assume valori da -inf a 1:

$$R2 = 1 - \frac{{}^{1}/_{n}\sum_{n}(y_n - \hat{y}_n)^2}{{}^{1}/_{n}\sum_{n}(y_n - \bar{y}_n)^2}$$

Osservare che tutte queste metriche sono tutte del tipo *"greater is better"*.

Training

Nella fase di training si vanno ad allenare, uno per uno, i vari modelli scelti su, ognuno dei dataset precedentemente descritti. Come nel caso della classificazione, al fine di ottenere il miglior setting possibile per una determinata coppia modello-dataset, si sono effettuati diversi training facendo variare i parametri di input, andando poi a selezionare il modello migliore.

La scelta del modello migliore avviene, anche in questo caso, tramite il calcolo di un *"total_score"*. Il setting di una coppia modello-dataset con total_score maggiore verrà scelto come migliore. Come total_score abbiamo scelto di utilizzare una somma pesata dei vari valori delle 4 metriche scelte; questo poiché ogni metrica ha dei range di valori diversi, e la stessa differenza tra due valori di due metriche diverso ha un "peso" molto diverso. Si è deciso di applicare una radice al -MSE in quanto si tratta di un errore quadratico.

I pesi sono stati scelti in modo tale da dare lo stesso peso alle variazioni dei valori di una determinata metrica, e ciò è stato fatto sulla base di un insieme di campioni empirici ottenuti da vari training, per cercare di avvinarsi nel miglior modo possibile a dei pesi equi (ogni metrica dovrebbe avere la stessa influenza sul total_score, altrimenti una metrica potrebbe prevaricare sulle altre e rendere le altre inutili nella scelta del modello migliore).

In particolare, analizzando più dataset e su questi andando ad analizzare tutti i modelli considerati si è visto che:

- $\frac{\sqrt{|-MSE|}}{-MAE} \sim \frac{3}{2}\ per\ cui\ se\ \sqrt{|-MSE|}\ ha\ peso\ 2\ allora - MAE\ ha\ peso\ 3$
- $\frac{\sqrt{|-MSE|}}{EA} \sim 10\ per\ cui\ se\ \sqrt{|-MSE|}\ ha\ peso\ 2\ allora\ EA\ ha\ peso\ 20$
- $EA \sim R2\ quindi\ \frac{EA}{R2} \sim 1\ ne\ consegue\ che\ EA\ e\ R2\ hanno\ lo\ stesso\ peso$

La formula finale con gli annessi pesi è data da:

$$TotalScore = w_{EA} * EA + w_{R2} * R2 + w_{-MAE} * -MAE + w_{-MSE} * -\sqrt{|-MSE|}$$

Nel nostro caso i pesi scelti sono: $w_{EA} = 20,\ w_{R2} = 20,\ w_{-MAE} = 3,\ w_{-MSE} = 2$, quindi:

$$TotalScore = 20 * EA + 20 * R2 + 3 * -MAE + 2 * -\sqrt{|-MSE|}$$

Andiamo ora ad analizzare come abbiamo fatto variare i parametri di input dei vari modelli nella fase di training:

- CART Regressor: non ci sono parametri di input significativi che sono stati fatti variare;
- Random Forest Regressor: i parametri di input sono il numero di alberi nella foresta, il quale varia in un range compreso tra 5 e 20, e il numero di features utilizzate per la creazione di ogni albero, che varia in un range compreso tra 1 e il numero totale di features;
- KNR: l'unico parametro di input è la dimensione k del vicinato, fatto variare in un range compreso tra 3 e 20;
- SVR: i vari parametri inseriti, utilizzati in funzione del nucleo scelto, sono:
    - Kernel, scelto tra "linear", "poly", "rbf", "sigmoid";
    - C, scelto in un range logaritmo compreso tra 10^{-2} e 10^{2};
    - Gamma, scelto in un range logaritmico compreso tra 10^{-5} e 1;
    - Degree, scelto tra 2 e 3;
    - Coef0, scelto costantemente uguale a 0.

La scelta di tali parametri è stata presa sulla base di documentazione di altri test di training trovati online e della teoria studiata in merito.

Da un punto di vista implementativo, la fase di training viene svolta all'interno del main del file *"training.py"*, nel quale avvengono due fasi principali:

3. Import e modifica del dataset scelto dal file csv, tramite i metodi *"resultAnalysis.case_full_dataset"* per il caso di dataset completi con eventuali strategie di normalizzazione o standardizzazione, e *"resultAnalysis.case_NaN_dataset"* per il caso di dataset con dati mancanti (5%, 10% o 15% di NaN), con strategia di filling scelta tra "mode", "mean", "median" e "eliminate_row".
4. Training di ogni modello per il particolare dataset scelto, tramite il metodo *"training.training"*, in cui ogni modello viene ciclato su tutti i possibili input. I setting migliori, scelti tramite il total_score, vengono salvati in dei file di setting, contenuti nella cartella *"best_model_settings/regression_model_settings"*, insieme ai valori delle metriche ottenuti.

Osserviamo che, per poter usare SVR nel dataset "energy", è stato necessario decorarlo con la classe *"sklearn.multiOutput.MultiOutputRegressor"*, in quanto tale dataset presenta più label.

Anche per il caso della regressione, il training unitario di un singolo setting per un modello viene effettuato utilizzando la tecnica del K-Fold-Cross-Validation.

Risultati ottenuti

La fase di analisi dei risultati viene effettuata all'interno del main del file *"resultAnalysis.py"*. In particolare, vengono letti tutti i vari file di setting per poter creare delle tabelle riassuntive dei risultati ottenuti. Ogni tabella contiene sulle righe i vari modelli usati, e sulle colonne le varie metriche utilizzate. Inoltre, ogni cella assume una gradazione di verde proporzionale al valore contenuto: verde più scuro corrisponde a valori più alti. Infine, viene realizzata una tabella finale riassuntiva contenete il miglior modello, scelto sempre sulla base del total_score, per ogni dataset, e modifica di quest'ultimo, analizzato.

Ecco alcuni risultati ottenuti:

	Dataset: airfoil - Strategy: mean - Perc. of NaN: 15% - Best model: RFRegressor			
	explained_variance	neg_mean_absolute_err	neg_mean_squared_err	r2
CARTRegressor	0.419405	-3.6053	-27.1929	0.41442
KNR	0.0716827	-5.20271	-43.9485	0.059992
RANDFORESTRegressor	0.665638	-2.7775	-15.7632	0.662092
SVR	0.206572	-4.78143	-37.571	0.199446

In questa tabella si deduce che Random Forest Regressor è il modello migliore rispetto ogni metrica analizzata, in quanto le celle relative alla sua riga sono le più scure.

	Dataset: energy - Strategy: eliminate_row - Perc. of NaN: 5% - Best model: SVR			
	explained_variance	neg_mean_absolute_err	neg_mean_squared_err	r2
CARTRegressor	0.964984	-0.86832	-3.24282	0.963879
KNR	0.927435	-2.07191	-6.88022	0.925707
RANDFORESTRegressor	0.978718	-0.773593	-1.93622	0.978399
SVR	0.981967	-0.865255	-1.68285	0.981656

Invece, in questo caso, il modello migliore è SVR, nonostante abbia un valore di -MAE inferiore a quello di Random Forest Regressor.

A questo punto possiamo mostrare anche la tabella riassuntiva per la regressione:

Summary table regression

regression	airfoil	auto	power_plant	compressive_strength	energy
full	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	SVR
normalize	RANDFORESTRegressor	SVR	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor
standardize	RANDFORESTRegressor	SVR	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor
5.0%_eliminate_row	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	SVR
5.0%_mean	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor
5.0%_median	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor
10.0%_eliminate_row	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor
10.0%_mean	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor
10.0%_median	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor
15.0%_eliminate_row	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor
15.0%_mean	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor
15.0%_median	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor	RANDFORESTRegressor

Osservando la tabella, si nota facilmente che quasi in tutti i dataset e in quasi tutte le strategie, il modello migliore risulta essere Random Forest; in pochissimi casi appare SVR.

Sviluppi futuri

Gli sviluppi futuri di questo progetto sono molteplici.

Nelle nuove versioni è interessante considerare nuove metriche per valutare i modelli, aggiungere nuovi Dataset e scegliere altre formule per valutare il “Best Model”.

E’ possibile, inoltre, migliorare la funzione di training su tutti i modelli parallelizzandola utilizzando molteplici thread; per esempio si può implementare un thread per ogni modello considerato.

Attualmente, le strategie utilizzate per fare il filling dei dataset con dati mancanti sono standard della libreria imputer e come detto precedentemente esse sono “eliminate_row”, ”median”, ”mean”, e “mode” . Per migliorare il programma, nelle future release, è possibile considerare nuove strategie di filling che dipendono dal modello scelto, in modo tale che la strategia di filling migliori significativamente le metriche del modello.

Riferimenti

[1] Uci Machine Learning Repository:
http://archive.ics.uci.edu/ml/datasets.html
[2] Kaggle Datasets:
https://www.kaggle.com/datasets
[3] Open datasets released by the U.S. government:
https://www.data.gov/
[4] Tour of machine learning algorithms:
https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/
[5] Metrics evaluate machine learning python:
https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/
[6] Scikit-learn metrics for classification, regression and clusterization:
http://scikit-learn.org/stable/modules/model_evaluation.html
[7] Normalizzazione e standardizzazione :
https://machinelearningmastery.com/scale-machine-learning-data-scratch-python/
[8] One hot encoding:
https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/
[9 Confusion matrix:
https://rasbt.github.io/mlxtend/user_guide/evaluate/confusion_matrix/
[10] Multiclass vs multilabel:
https://en.wikipedia.org/wiki/Multiclass_classification/
[11] Imputer scikit-learn:
https://machinelearningmastery.com/handle-missing-data-python/
[12] Imputing missing values before building an estimator:
http://scikit-learn.org/0.18/auto_examples/missing_values.html
[13] Imputation of missing values:
http://scikit-learn.org/dev/modules/impute.html
[14] Imputation of missing values:
http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Imputer.html