Davide Guidobene 877150 01/14/2022

Report Progetto Data and Web Mining

Il progetto è un task di classificazione basato sulla competizione Kaggle ZillowPrize (https://www.kaggle.com/c/zillow-prize-1/overview/evaluation) tenutasi nel 2017. L'obiettivo della competizione era quello di predire il logerror delle "Zestimate" (stime dei valori di un dataset di immobili), rispetto al prezzo di vendita effettivo:

$$\mathbf{logerror} = \mathbf{log(Zestimate)} - \mathbf{log(SalePrice)} \quad (1)$$

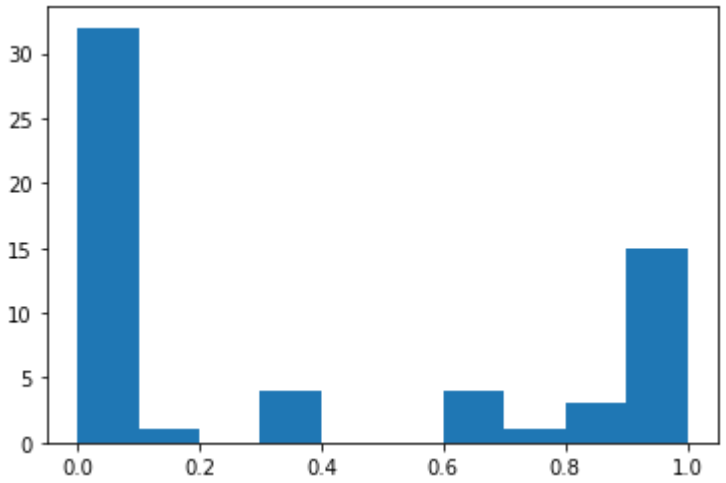
Task

Il task originale si basava sul minimizzare il Mean Absolute Error della predizione del logerror, ma per il progetto universitario ci è stata lasciata maggior flessibilità e io ho deciso di usare il Mean Squared Error che permette di effettuare il training di diversi modelli della libreria scikit-learn (per esempio le random forest: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html).

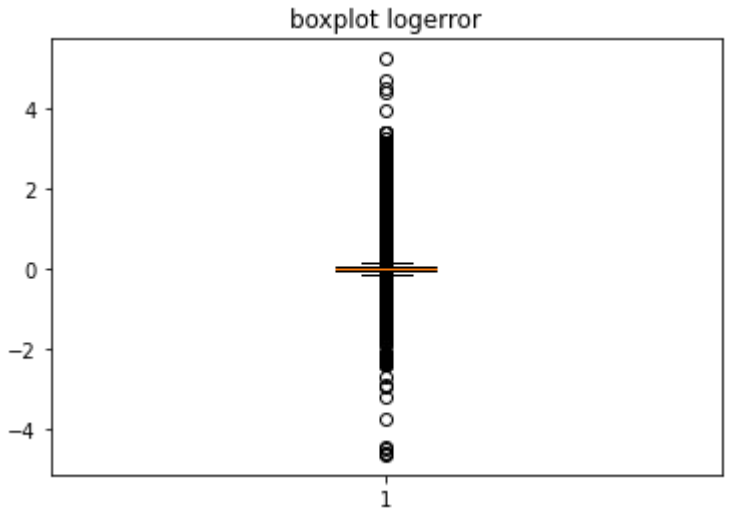
Data Preprocessing

Durante questa fase ho:

- trasformato la transactiondate in un formato numerico (numero di giorni a partire dalla prima transaction nel dataset: 2016/01/01)
- creato una nuova feature "countalreadysold" che conta quante volte era già stata venduta una casa prima della transazione rappresentata in quella riga (l'idea è che probabilmente la Zestimate sarà più accurata avendo già dei dati sulla vendita di quella casa)
- gestito alcune variabili particolari in cui NaN rappresentava 0 o False
- eliminato le variabili con una densità di NaN superiore a una certa tolleranza NAN_TOLLERANCE (che ho posto a 0.6 dopo aver studiato l'istogramma delle densità di NaN tra le varie features)



- diviso il dataset in variabili
    - binarie
    - categoriali (con n>2)
    - numeriche (ordinali e continue)
- diviso il dataset in train, validation e test
- Nota: queste divisioni del dataset non sono state fatte splittando il dataset in altre variabili ma creando delle apposite funzioni che restituissero la porzione del dataframe appropriata (in questo modo mi basta aggiornare solo il dataframe principale se voglio fare delle modifiche)
- Ho gestito le variabili binarie transformandole in 0 e 1
- Ho gestito i NaN delle variabili binarie, ordinali e continue sostituendo il valore mancante con la moda per le prime 2 e con la media per la terza. Ho poi aggiunto una feature binaria isnan_ corrispondente a quella colonna (1 se la feature era precedentemente NaN, 0 altrimenti)
- Ho gestito le variabili categoriali con OneHotEncoding, siccome diverse features avevano molti valori distinti, ho trasformato tutti i valori tranne i 5 più frequenti in ‘other’ e se i 5 più frequenti in OTHER_COL_VALUE. Se questi 5 valori non raggiungevano neanche una densità pari a 1-NAN_TOLLERANCE (0.4) nella feature in questione, allora ho eliminato la colonna del tutto.
- Ho osservato la distribuzione dei valori di logerror con un boxplot



- Notiamo che ci sono molti outliers, tuttavia la distribuzione di questi outliers non è innaturale ma continua. Quindi ha senso tenerli per il task di predizione. Nota: ho comunque sperimento e provato a eseguire il notebook levando la maggior parte degli outliers dal train e non ho riscontrato differenze significative nei modelli. Il codice è stato rimosso perché ridondante e non utile al task di predizione.

Modelli usati

Prima di tutto ho stimato il MSE usando come predittore di logerror la media dei logerror del train e l'mse è 0.0338490579289059.

Ho usato 3 modelli di regressione. Per ciascuno di essi ho eseguito una recursive feature elimination with cross validation sul validation dataset utilizzando la funzione RFECV del pacchetto scikit-learn. Durante questo processo ho dovuto assumere i parametri su cui non avevo neancora fatto tuning. Per fare tuning ho usato la funzione HalvingGridSearch del pacchetto scikit-learn, è una funzione nuova che ottiene risultati paragonabili a quelli di GridSearch ma con performance eccezionalmente migliori.
Ora descriverò come ho gestito questi modelli:

- RandomForestRegressor
    - ho usato alberi fully grown e fatto tuning sul numero di stimatori facendo HalvingGridSearch per valori in range(100, 1001, 100)
    - durante RFECV ho assunto 100 come numero di stimatori per una questione di efficienza

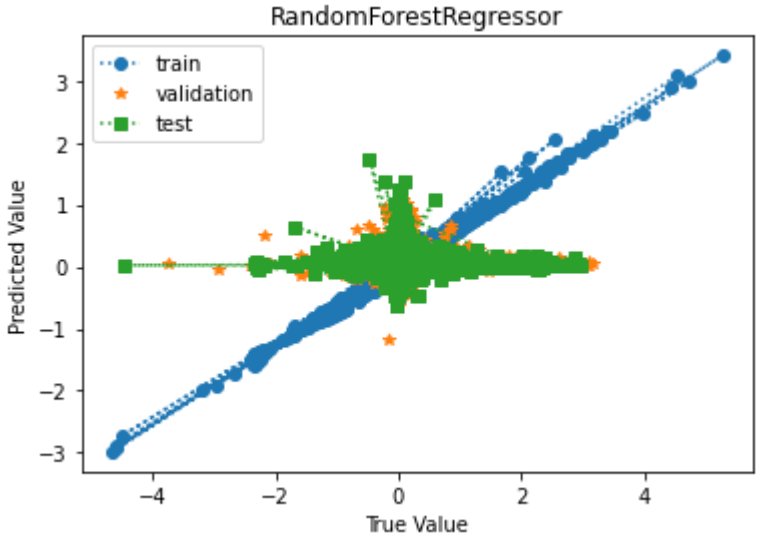
- AdaBoosting su base_estimator = DecisionTree
    - ho fatto tuning sul limite massimo di foglie e il numero di stimatori, facendo HalvingGridSearch rispettivamente in range(2, 100, 2) e in range(2, 100, 2). Avendo notato che il modello predilige valori bassi per entrambi non ho ritenuto necessario aumentare il range
- LinearRegression
    - siccome l'unico parametro su cui fare tuning sarebbe l'esponente delle features e siccome il dataframe corretto da PolynomialFeatures con d>=3 non stava in memoria ho deciso di trattare LinearRegressione con d=1 e con d=2 come 2 modelli separati

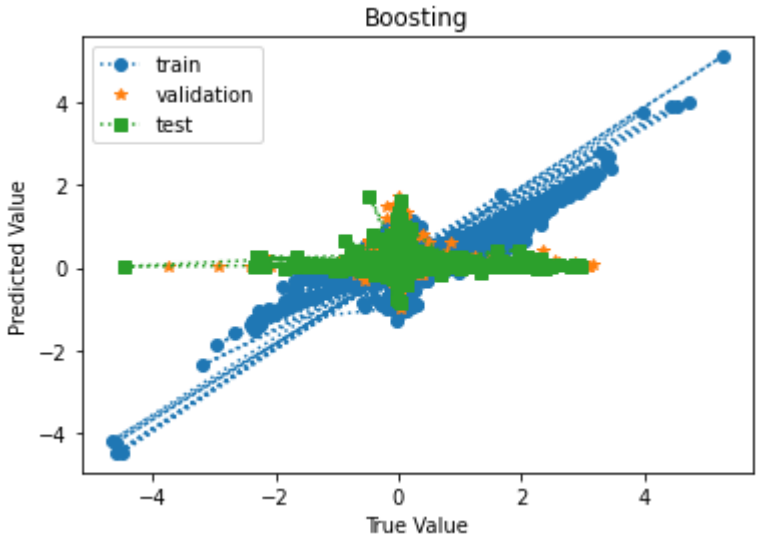
MSE dei modelli sul test:

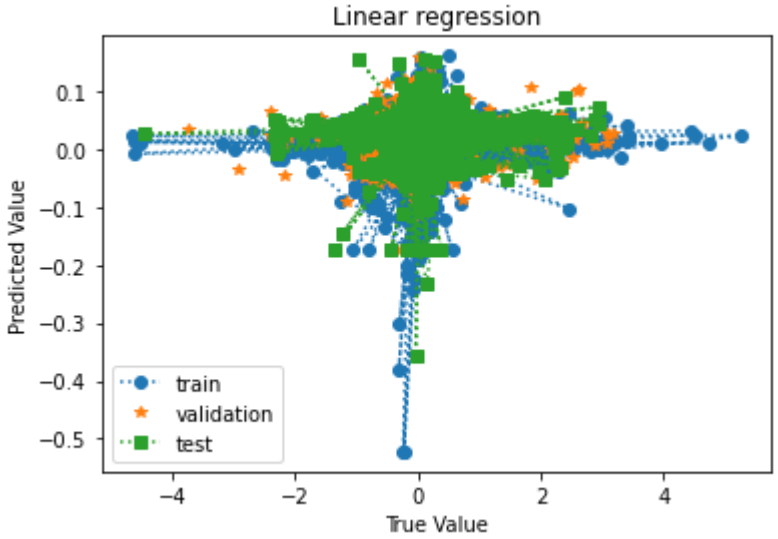
- RandomForest: 0.0370067611467574
- AdaBoosting con DecisionTree: 0.04567569606204479
- LinearRegression con d=1: 0.033665082149755 5
- LinearRegression con d=2: 0.2950100991616943

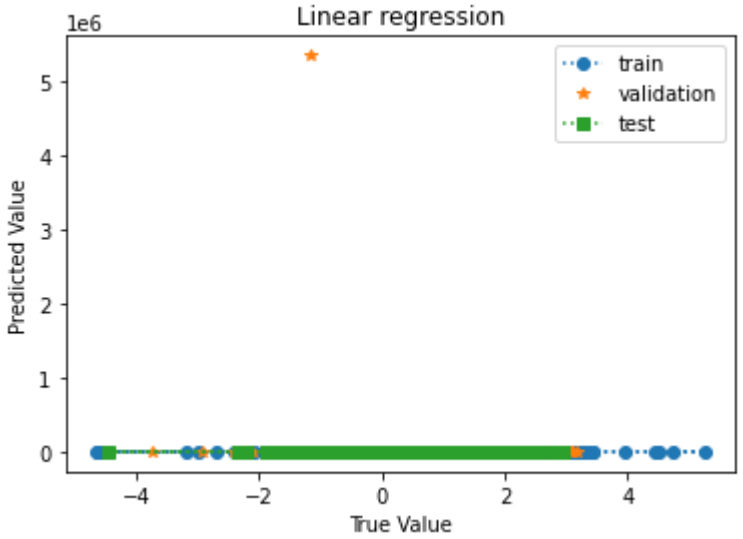
I modelli migliori risultano essere RandomForest e LinearRegression con d=1 Tuttavia nessuno migliora molto la predizione rispetto al fare la media di logerror sul test. Pertanto non sarebbero risultati particolarmente competitivi per una competizione reale.

Seguono i grafici delle predizioni dei 4 modelli



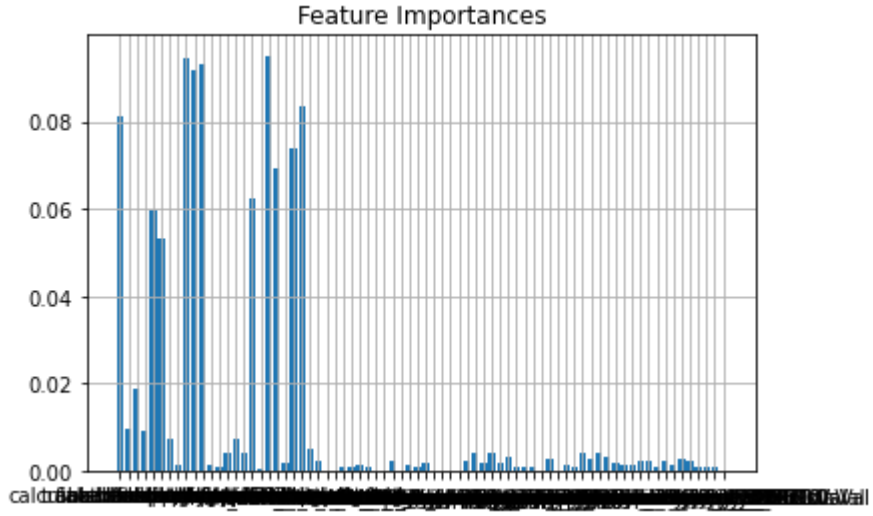

Boosting
train
validation
test
Predicted Value
True Value
4
2
0
−2
−4
−4
−2
0
2
4


Linear regression
train
validation
test
Predicted Value
True Value
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−4
−2
0
2
4


Linear regression
1e6
train
validation
test
Predicted Value
True Value
5
4
3
2
1
0
−4
−2
0
2
4

LinearRegressor con d=2 era il modello con l'accuracy su test peggiore e su questo grafico possiamo vedere perché: il modello resituisce delle previsioni molto fuori scala (troppa varianza).
Anche gli altri modelli non sono granche su test e validation, senza presentare una correlazione visibile tra true logerror e logerror predetto. Boosting e RandomForest mostrano questa correlazione per lo meno sul train, mentre linear regression neanche su quello.

Feature importance con RandomForest



Peso dato alle feature dalla random forest per la predizione di logerror.
Le features con un peso più alto sono quelle con importanza maggiore. Le etichette non sono leggibili in quanto troppo strette ma notiamo comunque che nessuna feature è particolarmente importante (anche se ce ne sono alcune molto più importanti di altre). Questo grafico si concilia bene con la bassa accuracy del modello: le features non ci sono feature che lo aiutano a spiegare bene il logerror e quindi il modello si limita a usarle per fittare il train.

Note finali

Ho provato a cercare gli outliers con ForestRegression e ho creato una funzione apposita. Tuttavia lo spazio in memoria richiesto per salvare la matrice di similarità era troppo pesante e perciò non è stato possibile aggiungere questo risultato al progetto.