Progetto Data Web Mining

Pastrello Marco 881679

1) **Analisi dei dati iniziali:**

Inizialmente ho scaricato i dati, quindi i 4 csv, e ho preso semplicemente quelli che secondo https://www.kaggle.com/c/zillow-prize-1/overview dovrebbero trattarsi del train, sia 2016 che 2017, e ho deciso di utilizzare solo questi come dati di analisi, quindi scartando tutto il resto delle properties, trattando questi come train e come test degli algoritmi che andrò ad applicare.
Osservando i dati ho capito come in realtà fosse possibile l'esistenza di più parceid nelle row del dataframe, perché quest'ultimo non rappresenta le singole proprietà ma i dati della vendita, quindi era possibile una ripetizione per quanto riguarda l'id delle proprietà. Questo mi ha portato a creare una colonna che rappresentasse, solo durante la gestione dei dati, unicamente ogni riga.

```
train_full['realID']= np.arange(0, len(train_full))
```

Successivamente ho analizzato i significati delle varie features da https://www.kaggle.com/c/zillow-prize 1/data?select=zillow_data_dictionary.xlsx e studiato i vari tipi con cui erano state memorizzate, dovendo averle tutte di tipo int o float per poter eseguire gli algoritmi.

2) **Gestione dati:**

Gestione features poco presenti:
Come prima cosa ho cercato di eliminare le features con un numero di valore nullo che superavano il 90% perché mi sarebbe risultato difficile predirne il valore mancante non avendo molti dati completi (come nel caso di buildingclasstypeid, finishedsquarefeet13, storytypeid, architecturalstyletypeid, decktypeid, finishedsquarefeet15, finishedsquarefeet50, finishedfloor1squarefeet, architecturalstyletypeid, typeconstructiontypeid, yardbuildingsqft17) o essendo feature poco presenti risultare poco utili per la predizioni o aggiungere errori (come nel caso di fireplaceflag, decktypeid, pooltypeid10, pooltypeid2, taxdelinquencyflag, poolsizesum, taxdelinquencyyear).

Gestione features non float o int:

Successivamente era necessario trasformare tutte le features nel tipo float o int, cioè propertycountylandusecode, propertyzoningdesc, transactiondate. Per le prime due avendo troppi valori unici non è possibile creare una colonna per ogni valore quindi trasformo le varie stringhe in un codice int diverso tramite un preprocessing.LabelEncoder(), con la possibilità di creare pattern errati portati da un ordinamento delle variabili che in realtà non esistevano, e per farlo prima sostituisco i NaN con un valore rappresentante per l'assenza del dato (in questo caso 0). Per transactiondate trasformo il valore in tre diverse colonne: una rappresentante l'anno, una il mese e una il giorno.

Gestione valori assenti:

Differenzio le features in categoriali, ordinali quantitative e numeriche (ordinali quantitative per significato della feature secondo https://www.kaggle.com/c/zillow-prize-1/data, categoriali per numero di valori unici possibili inferiore di quello della feature categoriale con questo valore più alto classificata categoriale per significato e numeriche per esclusione).

3) Scelta algoritmi:

Ho scelto di applicare i seguenti algoritmi:
· DecisionTreeRegressor: analizzando i dati e la variabile da predirre (logerror) ho pensato che applicare un decision tree visto la velocità di esecuzione ed essendo un algoritmo greedy (avendo un elevato numero d'istanze).
· BaggingRegressor e AdaBoostRegressor per poter gestire il bias o la varianza dell'albero per poter ottenere risultati migliori.
· RandomForestRegressor: per poter applicare un algoritmo solitamente dai risultati ottimi e confrontarlo con i precedenti, analizzando il confronto di risultati su diversi algoritmi che usano diversi alberi di decisione (in più ha un learning efficiente necessario visto l'elevato numero di istanze).

Per eseguire il tuning dei parametri decido di utilizzare la cross validation così da poter ottenere un risultato quanto più corretto possibile sui dati in possesso e dei valori provati differenti anche se di numero e dimensione limitati per questioni di tempo di esecuzione e spazio nella memoria.

4) Aggiunta nuove features:

Divido il dataframe in X (tutte le feature che non siano logerror e che non identificano l'istanza stessa (realID, parcelid)) e Y (logerror) e successivamente divido in train e test (train=75% test=25%)

Lo score è calcolato sulla base delle indicazioni del sito https://www.kaggle.com/c/zillow-prize-1/overview/evaluation.

Applico un DecisionTreeRegressor (con 5,10,50,100 foglie) e un RandomForestRegressor (con 10 alberi) prima di aggiungere le feature per studiarne la significatività delle aggiunte.

Aggiungo features basate sulla posizione:
· distanceFromCenter: distanza rispetto al centro, data la longitudine e la latitudine della propietà e delle varie County rappresentate dal fips (longitudine e latitudine rappresentativo del centro del County prese da https://www.gps-longitudine-latitudine.it/coordinate-gps-di-los-angeles , https://www.gps-longitudine-latitudine.it/coordinate-gps-di-orange , http://oralocale.timein.org/stati-uniti-damerica/ventura/map e trasformate nello stesso modo con cui è stato salvato nel csv (divisi per 1e6)), calcolo la differenza di longitudine e di latitudine
· NinCity: numero di propietà con lo stesso regionidcity

Aggiungo features basate sulle tasse:
· rProperty: rapporto del costo della propietà sul costo della terra (structuretaxvaluedollarcnt/landtaxvaluedollarcnt)
· AvgTaxRegionid: structuretaxvaluedollarcnt medio per regionid

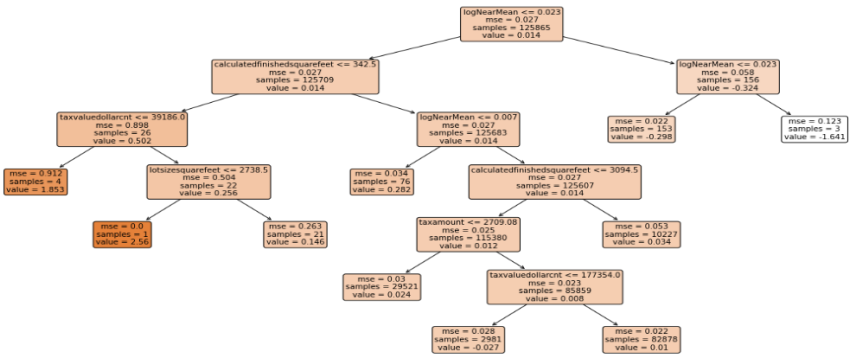
Aggiungo features basate sulla dimensione:
· ExtraArea: quantità di spazio extra medio in quel regionidcity (lotsizesquarefeet - calculatedfinishedsquarefeet)
· LivingArea: proporzione di area vivibile medio in quel regionidcity (calculatedfinishedsquarefeet/lotsizesquarefeet)

Aggiungo feature basato su il logerror:
Per aggiungere questa feature parto dal presupposto di avere dei dati con logerror già calcolati, questo implica una cold start nell'applicare l'algoritmo (inizialmente sarà più inefficiente o da applicare un altro algoritmo).
· logNearMean: media del logerror delle istanze precedenti (month minore) dello stesso anno e dello stesso County (fips) escludendo l'istanza stessa.

5) Analisi, tuning e confronto algoritmi:

DecisionTreeRegressor con 100 foglie:
La predizione sul test del DecisionTreeRegressor con 100 foglie ottiene un errore medio di 0.06810711000593345 dimostrandosi un ottimo algoritmo veloce.
Le istanze più corrette sono caratterizzate da un valore nella media minore per taxamount, taxvaluedollarcnt e maggiore per structuretaxvaluedollarcnt; quindi, sembrerebbero proprietà nella quale il valore della struttura stessa sia maggiore. Soprattutto queste istanze hanno un valore relativamente basso di logerror, dimostrandosi proprietà più facilmente predette dall'algoritmo utilizzato da Zillow.
Le istanze più errate sembrano avere, nella media, un maggior logerror mantenendo però un logNearMean simile a quello della media tra tutte le istanze (rendendoli possibili outliers), questo mi porta a capire che siano proprietà particolari avendo un elevato valore di NinCity seguito da un elevato valore di taxamount, structuretaxvaluedollarcnt, taxvaluedollarcnt, landtaxvaluedollarcnt, lotsizesquarefeet.

Bagging Algorithm su alberi con 10 foglie:
Analizzando i risultati dell'albero a 10 foglie decido, visto le dimensioni ridotte, di provare a crearne una rappresentazione grafica

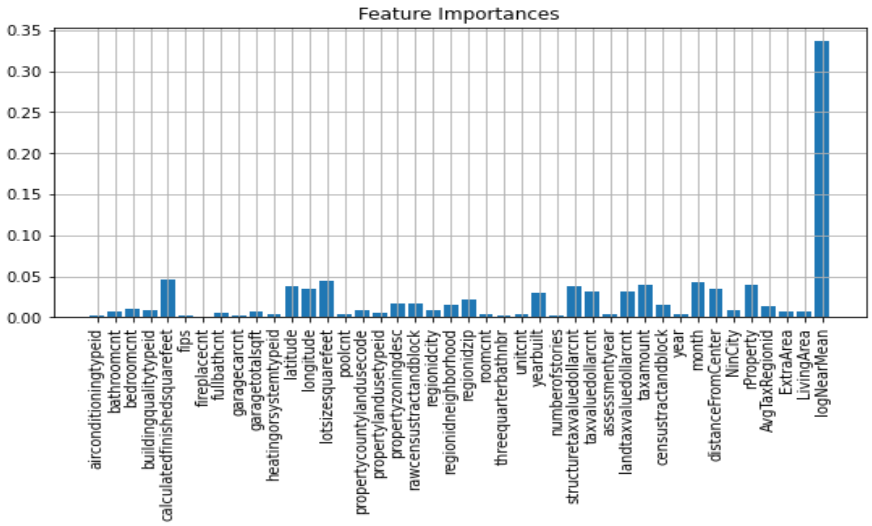


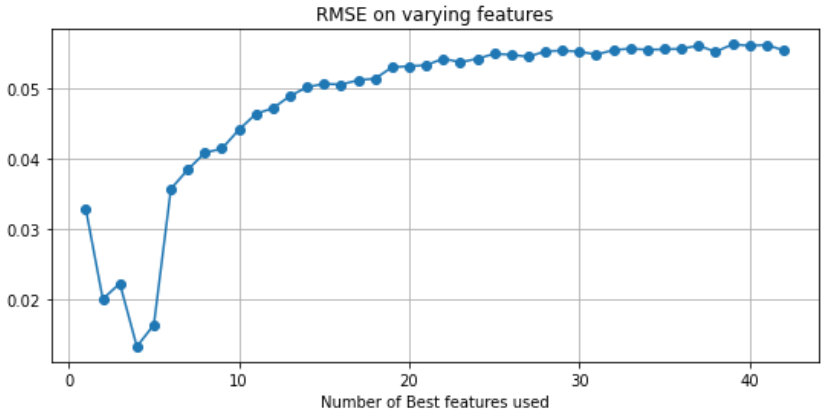
Essendo un albero poco cresciuto ha un Bias elevato, tuttavia aumentare la dimensione dell'albero implicava un tempo di esecuzione troppo elevato. Utilizzo la cross validation per effettuare il tuning del parametro rappresentante il numero di cicli eseguiti ottenendo i seguenti risultati.

Con n_estimators a 150 lo score diventa peggiore quindi utilizzo quello con 100 ed eseguo la predizione sul test ottenendo un errore medio di 0.06911187845355485 che è un risultato ottimo ma inferiore a quello del DecisionTreeRegressor con 100 foglie, probabilmente perché un albero con così pochi features significative utilizzate come quello con 10 foglie predice molto peggio anche dopo il miglioramento dell'applicazione del Bagging Algorithm.
Le istanze più corrette sembrano avere, nella media, un logerror simile a logNearMean, essendo logNearMean la feature più usata da un albero con solo 10 foglie questo rende queste propietà particolarmente corrette nella predizione essendo punti nella media e l'applicazione dell'algoritmo non riesce a migliorare di molto la situazione.
Le istanze con più errore sembrano avere, nella media, un più elevato numero di roomcnt, taxvaluedollarcnt, taxamount, AvgTaxRegionid, lotsizesquarefeet. In più queste sono le propietà che hanno un logerror diverso da logNearMean rendendo questi punti dei possibili outliers rispetto a logerror.

AdaBoost su albero con 100 foglie:
Avendo ottenuto ottimi risultati dall'albero con 100 foglie ho provato l'applicazione dell'AdaBoost su quest'ultimo; quindi, ho selezionato come iper parametro 5 per l'ottenimento di un risultato migliore.
Nel test si ottiene un errore medio di 0.07583176827123524, il risultato peggiore tra tutti. Questo risultato è dovuto alla maggior propensione nell'ottenere i risultati migliori con alberi decisionali piccoli che con poca varianza predicono sempre risultati vicini tra loro e introducendo l'algoritmo di AdaBoost non si introduce un miglioramento visto che quando il Decision Tree sbaglia vuol dire che quei valori sono fuori scala e non è sufficiente l'utilizzo di quest'ultimo per predirne il vero risultato.

RandomForestRegressor:
Seleziono non il migliore ma quello con 50 alberi per rapporto velocità di esecuzione e risultato, successivamente analizzo che features utilizza e con che importanza





Come si può notare dal grafico a sinistra la feature aggiunta logNearMean è quella più significativa superando molto le altre per importanza. Guardando questo risultato quindi mi chiedo se è possibile eliminare features inutili e quindi eseguo una dimensionality reduction applicando questo algoritmo prima sulla feature più importante, successivamente su un'altra e così via analizzandone i risultati (operazione effettuata su metà delle istanze per tempo e spazio in memoria).

Osservando il grafico a destra si può vedere come l'aggiunta della quinta variabile aggiunga dell'errore che successivamente continua a crescere, questo potrebbe essere dovuto a diversi fattori: l'aver introdotto errore nei dati cercando di rimpiazzare i NaN, le feature aggiunte portano alla creazione dei diversi alberi in maniera errata sviluppandosi su feature meno significative e che quindi cerca pattern non esistenti; invece, senza si esclude la possibilità di errore nella creazione.
Applicato il modello al test ottengo un risultato di 0.009517146214435989, questo perché essendo logNearMean una feature così significativa, la possibilità di allenare una foresta su quest'ultima e su poche altre, rende questo modello particolarmente efficiente, soprattutto con un elevato numero di istanze con logNearMean calcolabile (ovviando il problema della cold start attribuibile a questo tipo di feature).
A conferma di ciò le istanze più corrette hanno un minor lotsizesquarefeet rispetto alla media e il valore di logNearMean molto simile a quello di logerror.
Mentre le istanze più errate hanno un valore particolarmente grande di lotsizesquarefeet in media e il valore di logNearMean diverso da quello di logerror rendendo queste istanze delle outliers rispetto a quelle vicine.