Progetto
Data & Web mining

Zillow Prize: Zillow's Home Value Prediction (Zestimate)

Report Progetto

Prodotto da

Matricola	Nome studente
878169	Alessandro Gasparello
882339	Alessandro Simonato

Università
Ca'Foscari
Venezia

Dipartimento di scienze ambientali ed informatica

2021/2022

Introduzione

Nel seguente documento verranno descritte ed illustrate le fasi di sviluppo del progetto **Zillow Prize: Zillow's Home Value Prediction (Zestimate)**. Nello specifico saranno evidenziate e spiegate le decisioni prese, che hanno portato al completamento dello stesso. Saranno utilizzati diversi grafici per mostrare alcuni dei dati importanti che hanno portato alle decisioni finali prese.

Creazione del dataframe

Inizialmente è stato fatto l'import delle diverse librerie e moduli che sono stati utilizzati all'interno del progetto. Successivamente sono stati caricati i dataset e trasformati in data frame con l'utilizzo della libreria Pandas. Sempre grazie a tale libreria è stato poi fatta l'unione tra i due dataset contenenti diverse feature utili, quali "**logerror**" (la variabile risposta) o la "**transactiondate**" (data acquisto), così da avere tutte le istanze all'interno di un unico data frame.

Pulizia dei dati

Stampando il contenuto e il tipo delle varie feature contenute nel data frame appena costruito, è risultato che per la maggior parte erano di tipo **float64** mentre solo alcune erano di tipo **object**. Le prime non presentavano problemi nella gestione, mentre per quanto riguarda le seconde è stato necessario effettuare altre operazioni che saranno illustrate successivamente. Inizialmente abbiamo cercato di gestire subito la feature "**transactiondate**" andando a crearne di nuove, come la '**builded_buyed**' che mantiene un conteggio dei giorni trascorsi dal giorno in cui è stata effettuata la vendita della casa all'anno in cui è stata effettivamente costruita (l'anno di costruzione viene considerato completo per ogni casa, quindi 365 giorni).
Questa nuova feature ci ha consentito di andare poi a rimuovere le istanze con "**parcelid**" duplicati, ossia id assegnati a più vendite della stessa abitazione. Ci siamo interessati nel mantenere le case con il valore più alto della feature "**builded_buyed**", in quanto corrispondenti alla transazione più recente. Abbiamo pensato che transazioni più recenti avrebbero fatto riferimento a valori complessivi della casa più attuali.
Per la pulizia dei dati abbiamo deciso di cancellare tutte le feature che avevano un missing rate maggiore del **50%**. Per quanto riguarda invece le fea-

ture con una percentuale inferire al **50%** abbiamo deciso di correggere i valori mancanti con la media, nel caso di variabili numeriche, oppure con la mediana per variabili di tipo categoriale. Questo processo è stato effettuato nelle fasi più avanzate del progetto, ossia dopo l'aggiunta di eventuali nuove variabili.

Featrue engeneering

In quanto ad aggiunta di nuove feature, abbiamo deciso di inserirne alcune sulla base di quelle preesistenti all'interno del dataset e di gestirne alcune con maggiore attenzione. Inizialmente siamo andati a sistemare due feature, rispettivamente "**house_dim**" e "**censustractandblock**". Lo scopo di queste due featrue è quello di recuperare i valori mancanti da altre presenti nel dataset. In entrambi i casi avevamo notato che potevano essere recuperati tali valori da altre feature molto simili se non identiche.
Successivamente abbiamo aggiunto due nuove feature binarie "**missed_room**" e "**pre_house_dim**". Il loro scopo è quello di mantenere informazioni riguardo alla mancanza di dati per quanto riguarda la presenza dei garage oppure la dimensione totale della casa, prima che tale informazione sia rimossa dal dataframe oppure gestita.
Abbiamo poi deciso di gestire quelle che erano le tre feature 'problematiche' di tipo **object** già citate in precedenza. Per esse abbiamo utilizzato l'algoritmo **LabelEncoder()** così da poter associare ad ogni singolo elemento unico un id numerico, facilmente gestibile dai nostri futuri modelli. Non abbiamo potuto utilizzare **OneHotEncoder()** in quanto tutte e tre le feature possedevano un numero molto elevato di valori unici, andando anche a selezionare solamente quelli più frequenti.
Nell'ultima fase di questa sezione abbiamo poi creato alcune nuove feature. La prima feature creata è quella inerente ai beni di lusso. Il suo scopo è quello di mantenere un punteggio sulla base della presenza di alcune determinati oggetti che abbiamo considerato di lusso, quali la SPA, la piscina, il condizionatore oppure il camino. La seconda feature creata è la '**ValueProperty**' il cui scopo è quello di stimare il valore dell'abitazione in base alle tasse applicate.
Abbiamo aggiunto poi una serie di feature che mantengono informazioni inerenti alla media di altre variabili. Nello specifico "**avg_valuexzip**" e "**avg_taxzip**" che memorizzano il valore medio stimato della casa e il valore medio delle tasse sulla base dello zipcode associato ad una casa. Le seconde due create invece sono "**avg_day**" e "**avg_month**" che prendono in considerazione il "**logerror**" medio in base al giorno d'acquisto (all'interno di un

mese) oppure al mese d'acquisto (all'interno di un anno).
Infine, l'ultima feature aggiunta è "**room_count**" che mantiene un punteggio in base alla stima del numero di stanze presenti in una casa e la dimensione di tali.
Dopo aver rimosso eventuali feature non più utili e quindi non necessarie da passare ai modelli, abbiamo deciso di affidaci all'algoritmo **RFECV** che allenando dei modelli di **Random forest**, ci ha selezionato quelle che erano le feature più importanti, ossia che avrebbero fornito un contributo maggiore nelle predizioni finali che saremmo andati ad effettuare.

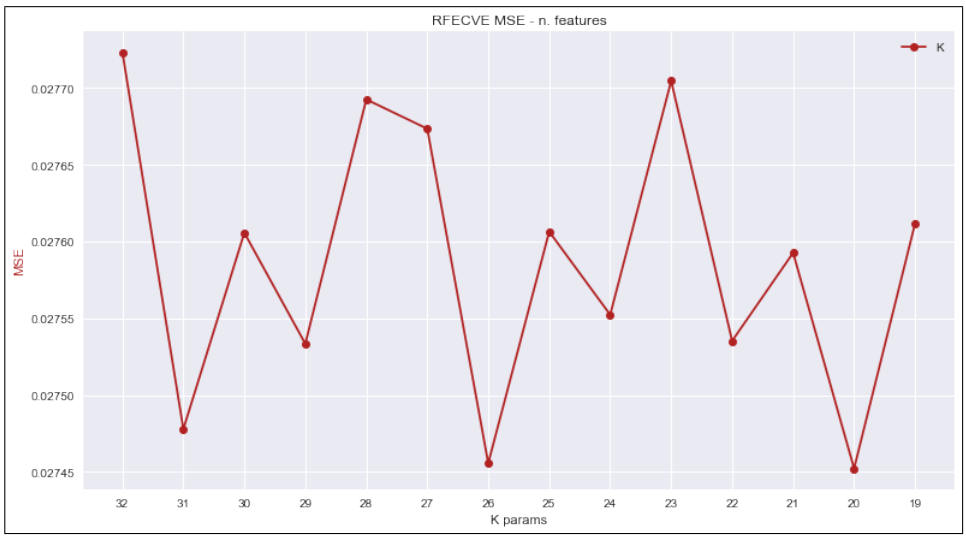

Figure 1: Grafico dei risultati ottenuto con **RFECV**. **MSE** al variare delle feature selezionate

Analisi dataframe post lavorazione dati

Terminata la fase di pulizia dei dati e la selezione delle feature, abbiamo effettuato la divisione in train e test dei dati ed abbiamo verificato come fossero distribuiti i dati della variabile risposta, ossia il "**logerror**". Dalla piccola analisi effettuata è risultato che la distribuzione era molto simile a quella di una normale, a parte per la presenza di alcuni valori anomali. Effettuando varie prove e rimuovendo infine tali valori dal train set abbiamo ottenuto valori migliori nelle predizioni finali, decidendo quindi di mantenere le modifiche apportate.

Creazione modelli di regressione e confronti

Come modelli di Regressione abbiamo deciso di utilizzare i seguenti : **Random forest**, **Regressione lineare**, **Knn**, **SVR** e **AdaBoost**. Per allenare i diversi modelli abbiamo deciso di utilizzare la funzione **GridSearchCV**, così da effettuare prima un tuning dei possibili parametri da utilizzare per ogni singolo modello e rilevare quelli migliori effettuando predizioni nei validation set. Abbiamo poi utilizzato i risultati ottenuti per andare ad allenare il modelli con i migliori parametri rilevati, ed effettuare così le predizioni finali sul test. Le statistiche ottenute sono poi state inserite in una tabella e confrontate.

Valid-Test scores	Mean Squared Error (MSE) Validation	Mean Squared Error (MSE) Test	mean exec time	best params
Models				
AdaBoost	0.006711	0.024627	0.578550	{'learning_rate': 0.01, 'n_estimators': 100}
Random forest	0.006674	0.024703	5.420105	{'max_depth': 40, 'n_estimators': 600}
Linear regression	0.006710	0.026247	0.003705	{}
Knn	0.006754	0.024684	13.103027	{'n_neighbors': 800}
SVR	0.006762	0.024653	23.172960	{'C': 0.03}

Figure 2: Tabella dei risultati finali ottenuti da ogni modello

La metrica per misurare l'accuratezza di un modello che abbiamo deciso di impiegare è lo **Mean Squared error**. Per ogni modello abbiamo mostrato quindi l'**MSE** ottenuto prima sul validation set e poi sul test set, il tempo medio d'esecuzione impiegato e i parametri che sono risultati essere i migliori per esso. Com'è facilmente notabile, il modello che ha offerto il più basso **MSE** sul test set è stato **AdaBoost**. Conseguentemente abbiamo deciso di utilizzarlo come riferimento per verificare quali fossero le istanze con un errore di predizione maggiore.

Analisi degli errori e conclusioni

In quest'ultima fase abbiamo quindi analizzato gli errori ottenuti nella predizione col modello migliore tra quelli testati. Inizialmente abbiamo provato a stamparci diversi scatterplot per vedere quale mostrasse maggiormente un pattern comune su cui il modello avesse più difficoltà nell'effettuare le predizioni, portandolo quindi a sbagliare. Tra tutti i grafici abbiamo notato che quello per la feature "**propertyzoningdesc**" mostra in due differenti distretti un particolare accumulo di errori più gravi rispetto alla media.
Nello specifico i due distretti maggiormente affetti da errori di predizione, com'è visibile anche nel grafico sottostante, sono quelli associati all'id "**1996**"

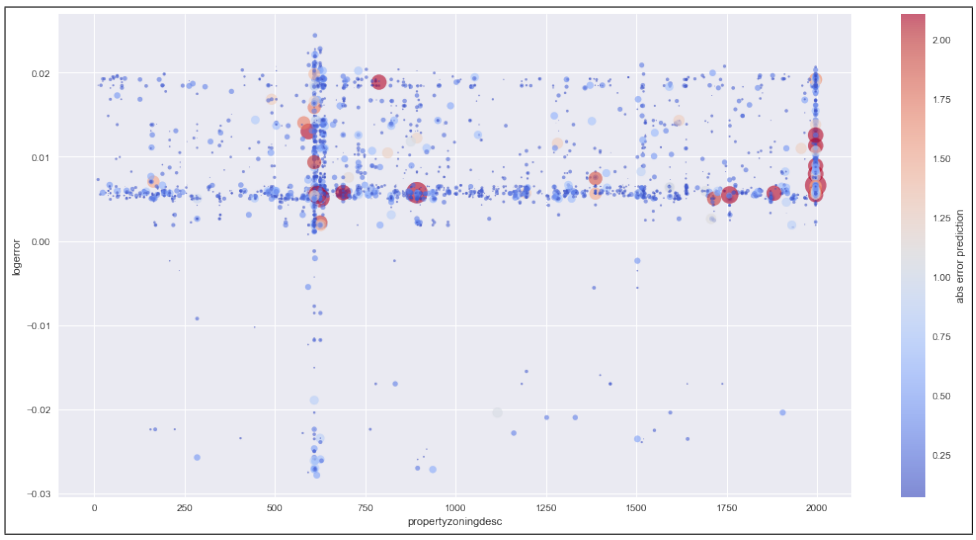

Figure 3: Scatterplot degli errori di predizione. La dimensione ed il colore di un punto identificano la quantità di errore nella predizione.

e nel range "**600 - 691**". Stampando tali istanze in un file csv, abbiamo notato che uno dei principali motivi che abbia portato ad un alto errore sopratutto in quelle zone è la presenza di molte feature affette da missing values, i cui valori sono quindi stati riempiti con la media e la mediana. Le due feature che probabilmente impattano maggiormente sull'errore di predizione sono "**builded_buyed**" (giorni passati tra ano costruzione e data ultimo acquisto) e "**structuretaxvaluedollarcnt**", in quanto risultavano tra le più importanti per effettuare le predizioni. Un'altro fattore interessante che può aver portato i modelli a commettere tali errori è dato dalla presenza di alcune istanze avanti valori anomali delle tasse sull'abitazione, ossia abitazioni aventi alti valori di tasse da dover pagare, nonostante le dimensioni contenute della casa. Questo potrebbe essere comunque dovuto alla zona d'appartenenza dell'abitazione, tuttavia è comunque una situazione che pensiamo sia giusto segnalare.
Infine, un'ultima particolarità che abbiamo notato, anche per quanto riguarda le istanze ben predette dal modello, è la tendenza ad effettuare la maggior parte degli errori soprattutto nelle istanze positive del "**logerror**". Nel grafico abbiamo escluso le istanze affette da poco errore, tuttavia controllando anche nel file csv era facilmente notabile come le istanze predette con un ridotto tasso d'errore erano tutte quelle associate ad un valore positivo di tale feature.
Nel complesso possiamo quindi ritenerci soddisfatti dei risultati ottenuti e dell'esperienza fatta.