Relazione progetto DWM 2021

Giovanni Costa - 880892 | Data and Web Mining | 15/06/2021

Indice

Features Engineering 2

Features importance and features selection 3

Costruzione dei modelli, analisi e conclusioni 5

In questo documento verranno: discusse le principali scelte implementative di progetto, descritte le analisi svolte, esposti i punti maggiormente problematici con le relative soluzioni e tratte le conclusioni. Per una descrizione più accurata del lavoro svolto si faccia riferimento al report jupyter notebook, come citato diverse volte all'interno di questo documento.

Features Engineering

Fin dalle prime analisi ci si è resi conto della complessità del dataset. Inoltre, non disponendo di strumenti hardware professionali per l'analisi del dato, sono stati adottati dei compromessi che in generale però non si pensa abbiano portato ad una degradazione della qualità dell'analisi svolta.
In particolare, i principali problemi sono stati:

1. Notevole numero di features e di istanze nel dataset
2. Missing value ratio molto elevato per la maggior parte delle features

Per far fronte al primo problema è stato necessario convertire tutte le variabili numeriche poste dalla libreria Python "Pandas" di default al tipo float64, in float32; così facendo sono stati risparmiati circa 600mb di RAM e sono state alleggerite tutte le varie operazioni sul dataset contenuto nel file "properties_2016". Inoltre, dopo una prima valutazione, ci si è resi conto che non era necessario operare su tutto l'insieme di dati citato precedentemente, avente quasi tre milioni di righe, ma solamente sulle istanze presenti anche nel dataset "train_2016_v2". Questo infatti è molto più ridotto con un conseguente vantaggio in termini di tempo per lo svolgimento di tutte le elaborazioni. Nello specifico, la dimensionalità complessiva una volta applicato l'inner join è stata 90275 righe e 60 colonne, più di due milioni e mezzo di righe in meno rispetto all'inizio. Per svolgere poi le operazioni di lettura e su memoria di massa, è stato deciso di utilizzare il formato Apache Parquet al posto del classico CSV, mediante la conversione del file originale fornito.

La strategia risolutiva per il secondo principale problema e per lo svolgimento del processo complessivo in generale è stata invece la seguente: si è deciso di dedicare molto del tempo impiegato alla progettazione del lavoro generale e degli steps da seguire, in modo da ridurre al minimo errori e intuizioni fuorvianti.
Si vuole evidenziare nello specifico come sia stata posta particolare attenzione alla manipolazione scrupolosa dei missing values ed alla piena comprensione dei diversi attributi presenti (semantica, tipo, valore, outliers) e della correlazione tra questi. Inizialmente quindi, dopo aver letto il file dictionary in formato .xlsx delle features e diversi dei notebook presenti su kaggle.com per avere opinioni di utenti terzi, si è passati all'analisi dei vari attributi, distinguendoli principalmente in categoriali e numerici.
Per quanto riguarda i dati numerici si è svolto uno studio sulla loro correlazione, che dato l'elevato numero di features riguardanti lo stesso topic, si è dedotto fosse molto probabile. Infatti, grazie a questa intuizione è stato possibile: eliminare i principali problemi di multicollinearità, rimpiazzare diversi missing value ed eliminare alcuni outliers.
Per alcune features categoriali e numeriche basate su posizione e tasse si è scelto di eseguire dei task di predizione dei valori missing tramite algoritmo KNN.

(le motivazioni e l'assunta correlazione tra variabili target e variabili predittive sono state dettagliate nello specifico all'interno del file jupyter notebook)

Si è voluto cercare di estrapolare il massimo dell'informazione disponibile dai dati presenti piuttosto che usare dei metodi di riempimento naif o rimuovere righe/colonne, anche perché tasse e posizione sono state ritenute informazioni fondamentali per la previsione del prezzo finale dell'edificio.

Per le restanti colonne invece si è proceduto al filling dei valori mancanti in maniera più basica, cioè utilizzando la moda per le features categoriali, e la mediana per le features numeriche. Si vuole far notare l'utilizzo della mediana per il riempimento rispetto a quello della media in quanto quest'ultima è più soggetta agli outliers, la cui presenza è stata confermata da diversi utenti su kaggle.com

Infine, una volta ottenuto un dataset abbastanza affidabile, si è potuto passare alla fase di creazione, sulla base delle colonne rimaste, di alcune features custom ipoteticamente utili per aumentare l'accuratezza della previsione richiesta in output.

Durante tutto il processo di features engineering, ma anche in altri punti:

- Sono stati utilizzati strumenti grafici per una migliore e più immediata comprensione dell'andamento dei dati.
- Si sono usati, dove necessario, i metodi della libreria "LabelEncoder" per la gestione di variabili categoriali di tipo stringa.
- È stato utilizzato lo "StandardScaler" per la standardizzazione statistica di alcune features in modo da uniformare la scala delle distanze euclidee usate nell'algoritmo KNN.
- Per la creazione dei modelli predittivi si è sempre utilizzata la tecnica del cross validation set per il tuning dei parametri in modo da evitare il riuso di stessi dati e per ridurre la possibilità di overfitting. Inoltre, si è sempre suddiviso il dataset in parte di train e in parte di test per rispettivamente allenamento e valutazione dei modelli.

Features importance and features selection

Il dataset una volta ultimato il processo di features engineering presenta una struttura di 90273 righe e 56 colonne. Come accennato precedentemente però alcune variabili sono di tipo categoriale e di conseguenza, perché siano interpretate correttamente, è d'obbligo codificarle come tali. È stato quindi fondamentale applicare a queste la tecnica del OneHotEncoding che però ha lo svantaggio di aumentare in modo esponenziale il numero di colonne, dovendone creare una per ogni valore distinto presente in quella d'origine. Qui è stato necessario adottare un compromesso per ottimizzare le elaborazioni: sono state rimosse dal dataset alcune delle variabili categoriali con molti valori distinti quali, ad esempio, "regionidzip", anche se queste sarebbero potute essere esplicative.

Si è passati poi all'allenamento di un modello di regressione sfruttante Gradient Boosting, usando l'implementazione messa a disposizione dalla libreria "XGBoost", che svolgesse il compito di selezionare le features di maggior impatto nella stima del logerror e che mostrasse l'importanza in termini numerici di ognuna di queste 72 features (numero di colonne dopo aver applicato il OneHotEncoding). Si è deciso di sfruttare questo modello perché molto potente e preciso, soprattutto se comparato all'uso di un singolo albero di decisione. Inoltre, generalmente, i predittori che utilizzano il metodo Gradient Boosting performano meglio di quelli che sfruttano Random Forest, l'altra tecnica di ensebling utilizzata all'interno del progetto. Il metodo Gradient Boosting per definizione ha come obiettivo la minimizzazione di una funzione di perdita e la caratteristica di costruire additivamente i vari alberi, anche se sono più proni all'overfitting.

Nell'immagine sottostante, raffigurante il grafico relativo al features importance score per le 40 colonne più importanti, è possibile notare che molte delle features custom aggiunte si sono dimostrate utili per la stima del logerror svolta dal modello di selezione con XGBoost.

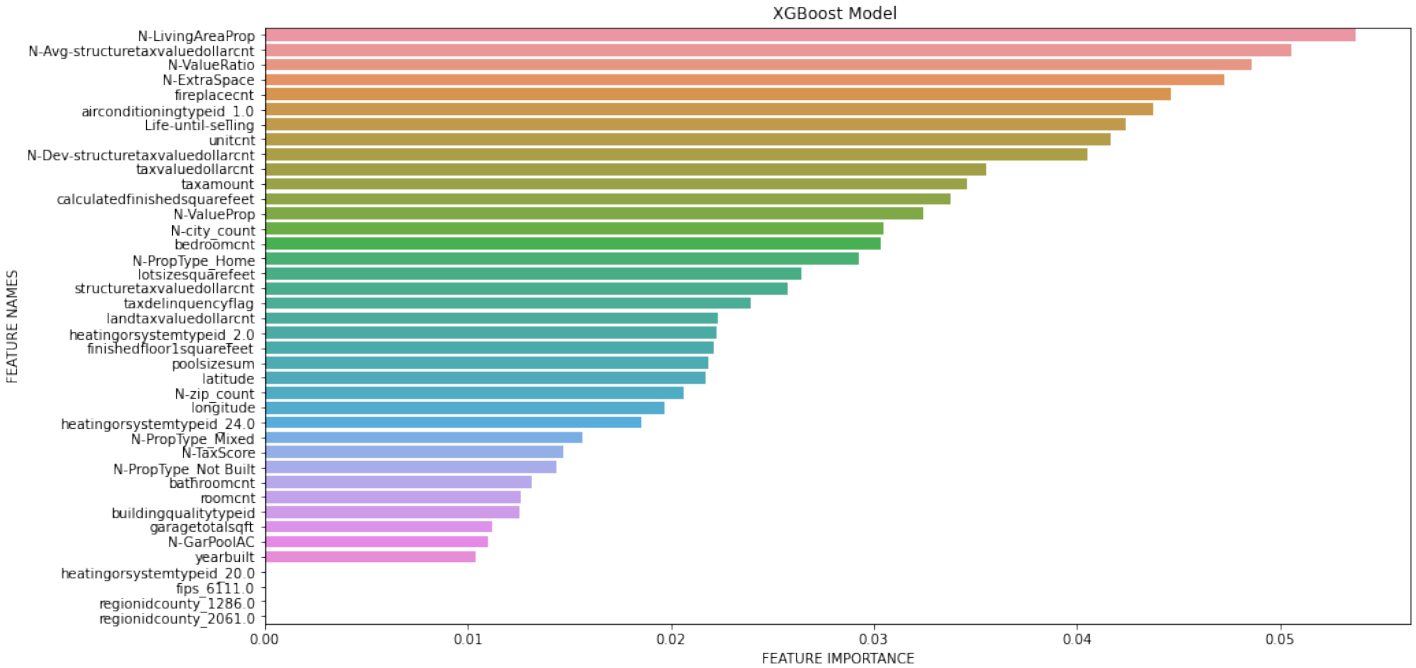


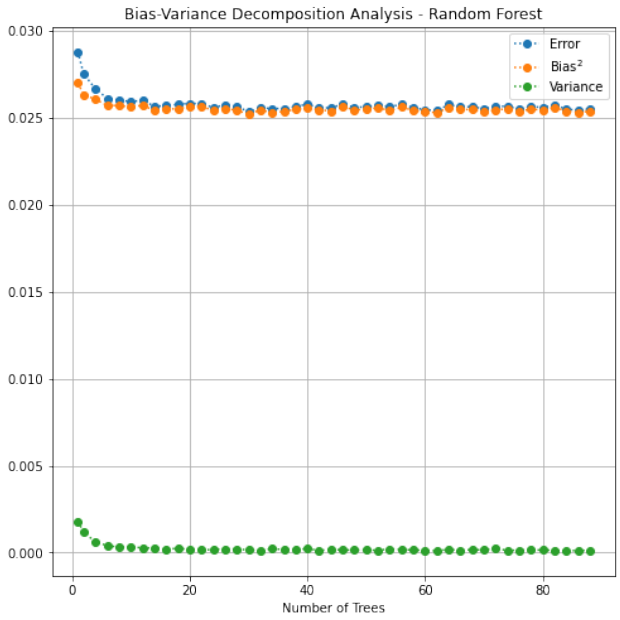
Figura 1: Grafico relativo agli importance score di 40 features

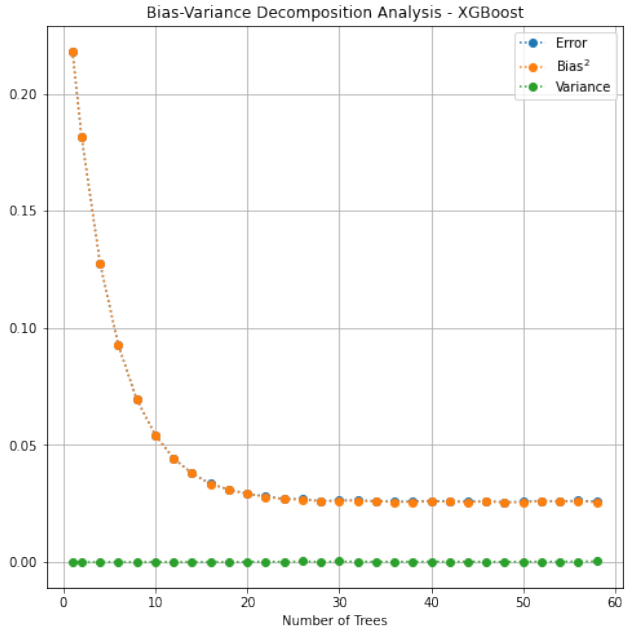
Si ritiene però che mediante un processo di rimozione ricorsiva delle features con cross validation set si possa arrivare ad un insieme di dati più espressivo e con minori problemi derivanti dalla dimensionalità elevata, tra cui multicollinearità e curse of dimentionality. Per lo svolgimento della rimozione ricorsiva si è deciso di impostare il parametro "min_features_to_select" della funzione a 10, perché lasciandolo al valore 1 di default il numero di features restituite si è visto fosse 3, considerato troppo basso per rappresentare l'informazione complessiva in modo corretto.

Costruzione dei modelli, analisi e conclusioni

Grazie ai passaggi precedentemente descritti si è arrivati ad avere un dataset contenente 90273 righe e 12 colonne da utilizzare per ricavare i due modelli finali sfruttanti rispettivamente Gradient Boosting e Random Forest. In particolare, durante la fase di tuning si è cercata la migliore combinazione tra 3 degli iperparametri principalmente usati in RandomForestRegressor ("n_estimators", "max_leaf_nodes" e "min_samples_leaf") e in XGBRegressor ("n_estimators", "max_leaves" e "learning_rate"), servendosi del metodo automatico messo a disposizione dalla libreria Sklearn "GridSearchCV".

(Si faccia riferimento al report jupyter notebook per maggiori dettagli)





Dai grafici in questione è evidente la capacità di queste due tecniche di ensembling nel ridurre in un caso la varianza e nell'altro la distorsione, in modo efficiente anche con pochi stimatori, riuscendo così a decrementare l'errore totale.

Si è infine svolta una comparazione tra i due regressori costruiti, utilizzando sia il punteggio di MSE ottenuto da entrambi nel testing set, sia valutando l'output restituito usando come input edifici con valore di logerror pessimo (item problematici), che ottimo. Il logerror si è visto seguire una distribuzione normale.

Modello	**MSE Score nel testing set**
Random Forest	0.0254
Gradient Boosting	0.0257

Figura 2: 7 item con logerror peggiore

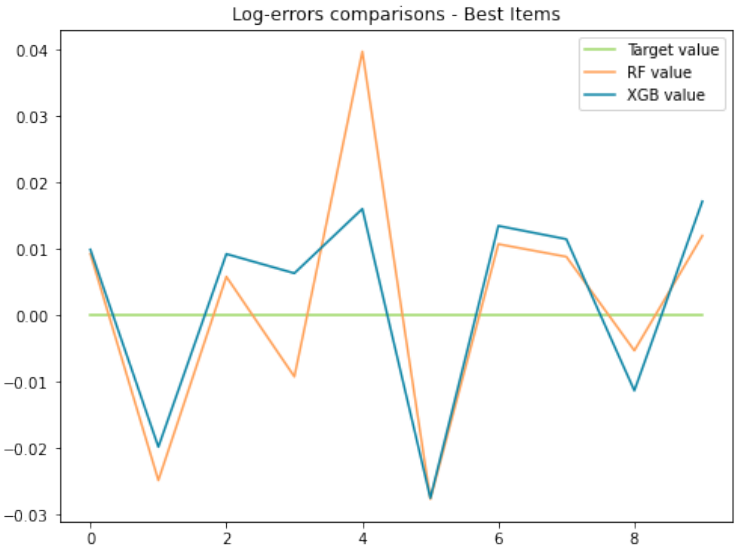


Figura 3: 10 item con logerror migliore

Dal bar chart relativo alle previsioni su item problematici si può notare che le stime dei due modelli finali creati sono molto distanti dal valore target calcolato dallo stimatore Zestimate. Probabilmente questo è dovuto alla presenta di molti valori missing o outliers, trattati con tecniche di analisi, recupero e correzione differenti dal team di Zestimate.
A priori però si ipotizza che gli item con logerror in valore assoluto elevato possano presentare anomalie o valori non standard per alcune features, in quanto proprio lo stimatore della competizione restituisce dei valori predetti abbastanza distanti dal log-SalePrice reale. Per verificare queste ipotesi si pensa sia necessario contattare degli esperti in ambito Data Science, il team di Zestimate e dei tecnici immobiliari.
Nel line chart inerente alle previsioni su item il cui logerror è nullo invece anche i due modelli creati restituiscono valori soddisfacienti: il discostamento rispetto al valore target di Zestimate è nell'ordine dei centesini.
In generale, anche nei grafici relativi a best items e worst items, non si notano grosse differenze in termini di accuratezza tra il modello con Random Forest e quello con Gradient Boosting, anche se il valore di MSE ottenuto nel testing set del primo modello è leggermente inferiore rispetto a quello restituito dal secondo.

Concludendo, grazie a questo task si è potuto provare su un dataset reale, esteso e complesso, la potenza di alcuni metodi di Machine Learning come Random Forest e Gradient Boosting, entrambi sfruttanti alberi di decisione ma utilizzanti intuizioni diverse. Tuttavia, come anticipato in precedenza, non si dispone di sufficienti elementi per preferire un modello rispetto all'altro in quanto entrambi ritornano previsioni abbastanza accurate ed in modo efficiente e il discostamento nel valore della metrica di valutazione per il testing set è molto bassa. Probabilmente, con una conoscenza più approfondita dei dati, del loro significato, del loro andamento reale e con l'aggiunta di informazioni esterne al dataset è possibile arrivare ad ottenere risultati ancora più precisi ed alla costruzione del modello di regressione ottimo.