Rental Listing Inquiries

Shobhit Behl
IMT2016024

Biswesh Mohapatra
IMT2016050

Aditya Sridhar Hegde
IMT2016054

***Abstract*—This project is part of the Machine Learning course taught in the International Institute of Information Technology (IIIT), Bangalore under Professor G. Srinivasaraghavan. In this report, we have described the dataset, our analysis and the approach adopted by our team.**

I. Introduction

The aim of the project was to predict the interest level shown by customers for new rental listings on the RentHop website based on data provided for past listings. Some of the features provided in the dataset include 'number of bathrooms', 'number of bedrooms', 'rental price', 'latitude', 'longitude' etc. using which we had to predict the 'interest level' which could take three possible values: 'high', 'medium' and 'low'; based on the number of inquiries on a listing.

II. Dataset

The training dataset consisted of 39481 rows while the test set consisted of 9871 rows, none of which contained any null values. However, the dataset was highly skewed with a majority of listings having 'low' interest levels as shown in Figure 1. Due to the large number of possible interactions between the attributes, a majority of our time was involved in data analysis and feature extraction.

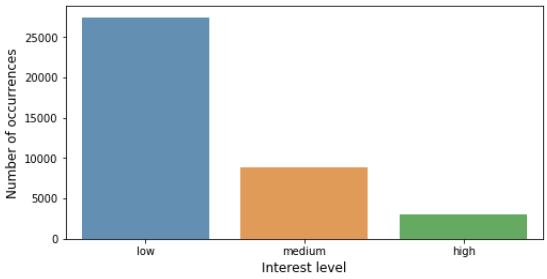

Fig. 1. Skew in training data

As shown in Figure 2, the correlation between the 'bathrooms' and 'bedrooms' attributes is comparatively high (~0.535), suggesting that they might not be independent. There is a high correlation between 'latitude' and 'longitude' too, though intuition suggests that they are independent and a high correlation is purely coincidental.

We now provide a brief analysis of each attribute.

A. Bathrooms

This attribute denoted the number of bathrooms in the apartment. More than 30000 rows had a value of 1 and all rows having a value greater than 4 had a low interest level. We couldn't observe any difference in the central tendency (mean) when grouped by the interest level suggesting that there wasn't a direct correlation to it.

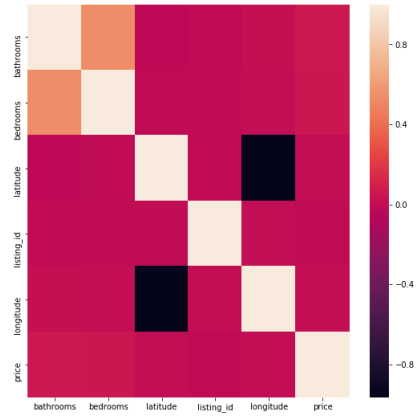

Fig. 2. Correlation between numeric attributes

B. Bedrooms

This attribute denoted the number of bedrooms in the apartment. The values were better distributed in this case but there wasn't an observable difference in the central tendency when grouped by the interest level.

C. Price

This attribute denoted the price for the apartment. As shown in Figure 3 there were quite a few outliers even after the removal of some extreme values (> 100000). We sought to remove outliers by restricting to data points having a Zscore in the range $[-3, 5)$, which translated to a price range of $[43 - 15046]$. Compared to the Inter-Quartile Range which gave an upper bound of 7000 for price, the Zscore seemed to give a more reasonable bound for removing outliers. After

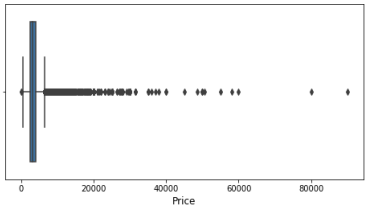

Fig. 3. Outliers in price attribute

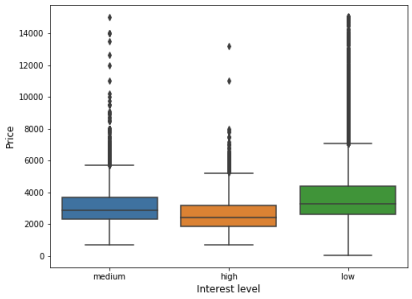

Fig. 4. Distribution of price conditioned on interest level

removing outliers, there seemed to be a direct relation between price and interest level as suggested by Figure 4 i.e. lower the price, greater the interest level.

D. Latitude and Longitude

These attributes represented the geographical location of the apartment. Both, the latitude and longitude attributes, had a few incorrect values which were completely isolated. For the purpose of our analysis, we approximated the top and bottom 1% records to the upper limit and lower limit respectively. As

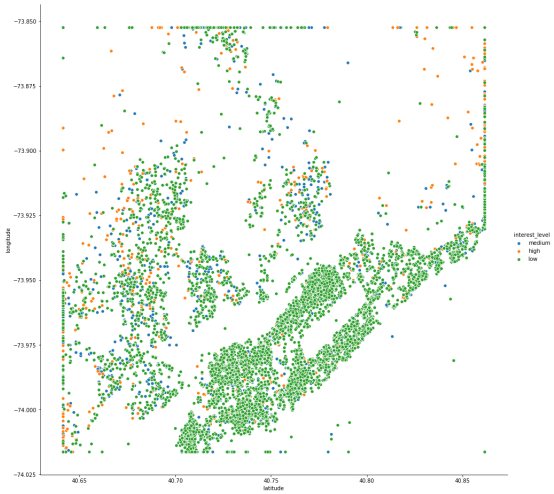

Fig. 5. Pairplot of cleaned latitude and longitude attributes

seen in Figure 5 there doesn't seem to be clusters of high or medium interest apartments, once again suggesting the absence of a direct correlation to interest level.

E. Photos

This attribute consisted of a list of URLs for the photos related to the apartment. Since we were asked not to use the images, the only feature we could extract from this attribute was the number of photos for each listing. The distribution of the number of photos didn't suggest any direct relation to interest level.

F. Created

This attribute denoted the date when the listing was created on the website. Some features we extracted from this attribute are 'day of creation', 'month of creation', 'year of creation' and 'number of days since creation'.

G. Feature

This attribute consisted of a list of strings which described the features of the apartment. We adopted different approaches for processing and using this attribute and hence a more detailed approach will be provided in Section III.

We also extracted the 'number of features' from this attribute but it didn't seem to show any direct relation to interest level upon analyzing its distribution.

H. Description

This attribute was a textual description of the apartment. Once again, we adopted different approaches for processing and using this attribute and hence a more detailed approach will be provided in Section III.

Some other features that we extracted from this attribute were 'number of words in upper case', 'ratio of number of upper case characters to that of lower case characters' and 'number of special characters'. Through these features we wished to capture changes in user opinion of a listing due to the formatting of textual content i.e. the possibility of text written completely in upper case or interspersed with special characters inducing a lower interest level in users.

Except for 'number of special characters' the other features didn't seem to show any direct correlation to interest level. Contrary to our intuition, 'number of special characters' did not have a linear relationship with interest level as shown in Table I. However there was a difference in the central

TABLE I
CENTRAL TENDENCIES OF NUMBER OF SPECIAL CHARACTERS IN DESCRIPTION

Interest Level	Mean	Standard Deviation
Low	14.94	15.62
Medium	18.29	15.78
High	16.83	17.76

tendencies when conditioned on interest level which suggested that it might be a useful feature.

I. Manager Id

This attribute denoted the id of the manager who put up the listing on the website. The training dataset consisted of 3292 unique values while the test set consisted of 2020 unique values, however, this attribute would truly be useful only if there was considerable overlap between the train and test datasets.

Fortunately, 1831 values were common to both datasets which would mean only 189 new values in the test set. Moreover, as shown in Figure 6, the number of listings put up by a manager in the test set is proportional to that in the train set which would make this an important categorical feature.

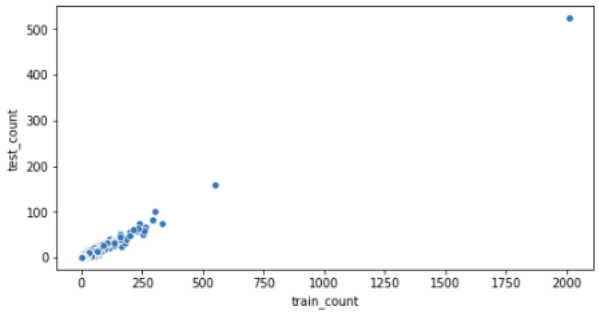

Fig. 6. Number of listings of each manager in train vs test datasets

J. Display Address

As the name suggests, this attribute denoted the display address of the apartment. There were 7644 unique values for this attribute in the training data. We were able to get it down to 4876 after processing the values. The transformations were mainly centred around converting all characters to lower case and removing special characters and stopwords.

K. Street Address

This attribute consisted of a more detailed address of the apartment compared to the 'display address' attribute. The training set consisted of 13451 unique values and we were able to bring it down to 10366 by processing the text. Our processing mainly involved converting all characters to lower case and removing special characters along with converting 'st.' and 'street' to 'st', 'west' to 'w', 'east' to 'e' etc,. The idea was to get similar values under the same notation e.g. some values used 'street' while others used 'st', so transforming 'street' to 'st' would introduce uniformity.

L. Building Id

In the system adopted by Renthop, a building was given a unique id and apartments could use this id to provide information about the building. This attribute thus denoted the id of the building to which the apartment belonged to and contained 6801 unique values in the training data.

While most values were alphanumeric strings, 6663 rows had a building id of 0 which could indicate that these rows had a missing building id.

III. Feature Engineering

The previous section talks about some basic derived features like 'number of photos', 'number of special characters in description', 'day of creation' etc,. In this section we list features derived from a combination of attributes or obtained after processing of data.

A. Distance

We extracted a feature called 'distance' from latitude and longitude by computing the Haversine distance from the city centre. Often the distance from the heart of the city plays an important role when looking for apartments and it was exactly this we wanted to capture through this attribute. To sanitize the attribute, since it contained quite a few outliers, we restricted rows to have a Zscore in the range $[-3, 3]$. As shown in Figure

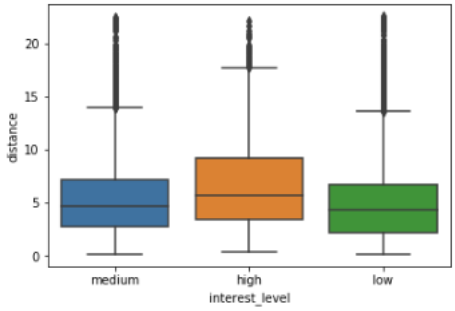

Fig. 7. Distribution of distance conditioned on interest level

7, there seems to be a correlation between distance and interest level since most apartments that are farther away from the centre of the city seem to have a higher interest level.

B. Total Rooms

Total Rooms is just the sum of the number of bedrooms and bathrooms. We felt that this attribute could be especially helpful for ensemble models since such models made use of comparison based methods leading to loss of features that occur as a combination of other features.

C. Price Per Room

In an attempt to extract features that were directly correlated to the interest level we computed the ratio of price to the number of bedrooms, bathrooms and total rooms (independently). The number of rooms is an important metric for apartments. We thus hoped to capture this, combined with the effect of the price attribute through these features. In order to account for zero values the exact ratio was calculated as

$$\text{Price per room} = \frac{price + 1}{rooms + 1}$$

As shown in Figure 8, the mean for low interest level apartments is significantly higher than that of the others while the mean for medium and high interest apartments are quite close, although this might be a consequence of the skewed dataset.

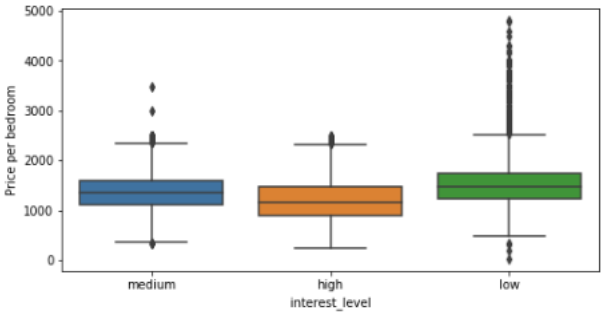

Fig. 8. Distribution of price per bedroom conditioned on interest level

D. Features

Most of the feature engineering for 'features' revolved around processing the text to extract key words.

One of our first approaches was to simply convert each feature, in the list of features for an apartment, to lower case followed by joining the words of the feature by an '_' and removing stopwords. We then used to sklearn's CountVectorizer class to create a one-hot encoding of the features while limiting the maximum number of columns to 150 (by keeping the most frequently occurring token).

We then sought to process each feature by

- Converting each character to lower case
- Replacing special characters with spaces
- Removing stopwords
- Using the nltk library's PorterStemmer for stemming

After the transformations above, we noticed that most of the important words could be extracted by considering unigrams and bigrams e.g. 'elevator' and 'no dogs'. We thus used sklearn's CountVectorizer for encoding unigrams and bigrams of the processed features while limiting the number of columns to 200 (by keeping the most frequently occurring tokens). We also used sklearn's TfidfVectorizer but it didn't lead to any improvement in our cross validation score.

We tuned hyper-parameters like max columns for one-hot encoding through cross-validation. It was surprising to see that the processed features didn't perform as well as our former, more simplistic approach.

E. Description

Our approach for processing the 'description' texts was similar to the one mentioned in III-D for features. Through cross-validation, we found that using only unigrams and limiting the max number of columns to 100 performed better than other approaches. Unfortunately, we weren't able to extract a significant improvement in our score through this feature. In fact it led to a deterioration in our cross-validation score by a significant amount.

We also extracted a few statistics as mentioned in II which turned out to marginally improve our cross-validation score.

F. Manager Ability

To work with the manager id attribute, which had high cardinality, we sought to generate a feature that accounts for the number of listings that received high, medium and low interest levels for each manager i.e. the ability of a manager to generate listings that receive high interest from customers.

We first computed the number of listings with different interest levels for each manager. This gave us a vector containing 3 components for each manager i.e. the number of high, medium and low interest listings. We then computed the dot product of this vector with a weight vector $\vec{w}$ to get a single numerical value for manager ability. For new managers in the test set, we took the average manager ability value.

Hyperparameter tuning involved setting the components of $\vec{w}$. Our first approach was to give a weight of 0 to low, 1 to medium and 2 to high interest apartments. Another approach we adopted was to weigh each interest level inversely to the number of occurrences i.e.

$$w_i = \frac{\text{total number of rows}}{\text{number of rows with interest level 'i'}}$$

The latter approach gave a better score during cross-validation.

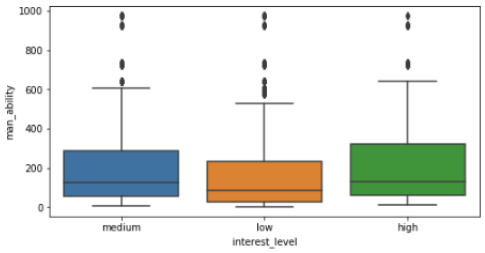

Fig. 9. Distribution of manager ability conditioned on interest level

G. Likelihood Estimates

We wished to compute likelihood estimates of the interest level conditioned on manager id, building id, price or a combination thereof. However, we weren't sure how to account for values not in the training set, especially for manager id and building id.

We discovered a method where we could compute these likelihood estimates against a subset of the training set and account for new values. The method adopted for likelihood estimates conditioned on manager id was as follows

1) Split the training data into parts A and B in a 80-20 ratio
2) For each distinct value of manager id in B, compute
$$likelihood_{ij} = \frac{n_{ij}}{n_j}$$
using A. Where n_{ij} is the number of listings with interest level 'i' by manager with id 'j' and n_j is the total number of listings by manager with id 'j'.
3) If the manager id in B did not exist in A a 'NaN' value was given for the likelihood.
4) Repeat the above procedure 5 times.

Due to step 3, where we gave 'NaN' to values only in B we were able to account for new values in the test set. As discussed before the number of managers unique to the test set were around 189 and these managers didn't have a lot of listings. Our approach above essentially ends up assigning 'NaN' to those ids that have relatively few listings since all listings of these managers will have to occur in B, which is just 20% of the dataset. The model thus ended up learning how to classify listings with 'NaN' values for likelihoods.

Using this approach, we computed likelihood estimates conditioned on 'manager id', 'building id', 'manager id and building id' and 'price and building id'. Likelihood estimates conditioned on 'manager id' performed extremely well during cross-validation and ended up leading to one of our best scores on the public leader board.

H. Label Encoding Categorical Attributes

We used sklearn's LabelEncoder for label encoding categorical columns like 'street address', 'display address', 'manager id' and 'building id'. We opted for label encoding over one-hot encoding since we were mainly using ensemble models and ensemble models tend to work better with label encoded values due to the comparison metrics used by it (splitting on a single attribute containing all the values leads to higher information gain than splitting on each one-hot column at separate nodes).

We tried label encoding the address attributes directly as well as after processing them as suggested in II, however, directly encoding them gave a better cross-validation score.

I. Processing High-Cardinality Categorical Columns

We adopted the method suggested in [1] for processing 'manager id'. We combined it with the approach used in III-G for handling new values in the test set. Unfortunately, this feature increased the error by a significant amount during cross-validation and so we didn't pursue the method for other attributes.

IV. Models

In section II and III we discussed about the attributes and features used for building models along with which features performed well during validation. In this section we give an overview of the model building approach adopted by our team.

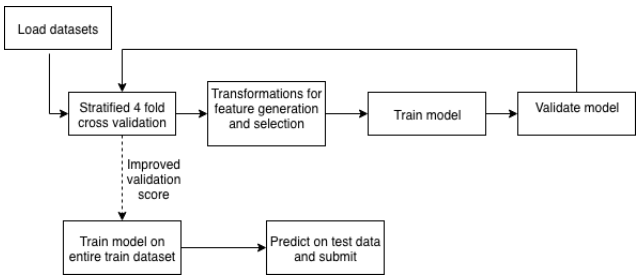

Fig. 10. Model building pipeline

As shown in Figure 10, we used Stratified 4-Fold cross-validation using sklearn's 'StratifiedKFold' class. We opted for stratified k-fold because of the skewed dataset. In each iteration of the 4 fold validation we ensured that all features were computed only from that iteration's training set and data from the validation set did not creep in when training models. We used the multi-class log-loss metric for

A. Logistic Regression

One of our first models was logistic regression on numerical attributes like price, number of bedrooms and bathrooms, latitude, longitude and subsets of these features. Since evaluation was based on multi-class log-loss metric we felt that logistic regression would provide a reasonably good score.

Among different subsets of numerical features, the model built using all the numerical features gave the best score on the public leader board (~0.725). Normalizing the attributes led to an improved score of 0.724.

B. Random Forest

Our first model with random forest was built using the same features as the best logistic regression model i.e. all numerical attributes. We got an average error of 0.682 during validation and a public leader board score of 0.656.

At this point, we were trying to extract important features and we opted for random forest since it works well on correlated attributes and also provides the feature importance information after training.

Through cross-validation we found that 'manager ability' and 'distance' didn't improve the score and features like 'price per bedroom' and 'total rooms' performed quite well. Using numerical features mentioned previously along with 'price per bedroom', 'price per total rooms' and 'total rooms' we were able to achieve an average error of 0.652 during validation and a public leader board score of 0.637.

We then tried building models using text attributes like 'description' and 'features' but due to the long training time we turned towards other models like LightGBM.

C. Gradient Boosting

Parallel to feature selection using Random Forests, we were building models using sklearn's GradientBoostingClassifier on features that performed well with Random Forest in order to see the best score we could extract from the feature set. We tuned the model's hyperparameters using cross-validation to obtain a value of

1) 0.08 for learning_rate
2) 0.8 for subsample
3) 100 for n_estimators
4) 8 for max_depth

We also observed that these hyperparameters gave the best validation score for most of the feature subsets we tried.

The best Gradient Boosting model gave an average error of 0.640 during validation and a public leader board score of 0.602. The features used for this model were the same as the ones used for the best RandomForest model.

D. LightGBM

When we started working with text attributes the training time was unreasonably high using sklearn's GradientBoosting-Classifer and RandomForestClassifier. We thus turned towards LightGBM which was an order of magnitude faster than the former. Moreover, the error obtained using LightGBM was close (or even less) than our previous models.

We built models using text features like the ones described in III-D and III-A along with numerical features selected from previous models.

Due to LightGBM's speed we were also able to perform a complete forward selection search to select the best subset of features by comparing the cross-validation scores. These features gave an average cross-validation error of 0.568 and an error of 0.5608 on the public leader board.

E. XGBoost

We used XGBoost to improve upon LightGBM's score. Unfortunately, we found that the features found through forward selection didn't work well in case of XGBoost. A slightly modified feature set gave an average error of 0.556 during cross-validation and an error of 0.553 on the public leader board.

We achieved a significant decrease in score by using the likelihood estimates described in III-G. Our best model consisted of

- Plain numerical attributes like 'bathrooms', 'bedrooms', 'latitude', 'longitude' and 'price'.
- Derived numerical features like 'total rooms', 'price per bedroom', 'price per bathroom' and 'price per room'.
- Number of features, number of photos and number of words in description
- Created month
- One hot encoding of features as described in III-D
- Direct encoding of manager id, building id, display address and street address
- Likelihood estimates on manager id

These features gave an average cross-validation error of 0.543 and a public leader board score of 0.538.

We tuned the hyperparameters using cross-validation to get a value of 5 for 'max_depth', 0.02 for 'eta' and 1700 for number of iterations. As shown in 11 the validation error stabilized after 1500 iterations.

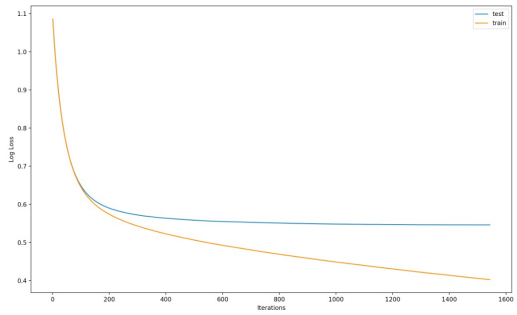

Fig. 11. Plot of train and test error with number of iterations

F. Stacking

Our approach for stacking was to build models that had low co-variance similar to the basic idea of ensemble models. We used

1) the best XGBoost model as the driver model with a weight of 0.8
2) a LightGBM model trained only on the 'features' attribute with a weight of 0.05
3) a LightGBM model trained only on 'description' based features with a weight of 0.05
4) a XGBoost model trained on equal number of high, medium and low interest listing to take care of the skewed dataset with a weight of 0.1

However, the cross-validation error increased to 1.120 with this model.

V. Conclusion

The model we submitted was the XGBoost model that gave the best score as described in IV. The scores obtained through this model are summarized in Table II. The confusion matrix

TABLE II
SUMMARY OF LOG-LOSS ERRORS

Type	Error
Cross-validation	0.54338
Public Leaderboard	0.53890
Private Leaderboard	0.53152

computed on validation set is given in Table III.

TABLE III
CONFUSION MATRIX ON VALIDATION DATA

	High	Low	Medium
High	242	163	364
Low	35	6378	459
Medium	160	1252	818

VI. CODE STRUCTURE

The code used during the project followed the Object Oriented Paradigm. It consisted of a base class called *Model* which contained basic functions such as select_training_features, select_test_features, sanitization and predict. The select_training_features and select_test_features methods were used for choosing the correct feature set for the model while sanitization was used for the data cleansing. All of our models inherited from 'Model' class. The inherited classes reflected their model type through their nomenclature such as *XGB_Model*, *LGBM_Model* etc. Each of these inherited class implemented a fit function to train the model on the given data.

Apart from these, our code included 2 more classes named *FeatureSelector* and *Transform*. *FeatureSelector* implemented all the methods concerning derived attributes along with the original attributes. *Transform* class implemented methods which worked on transforming the data set based on our observations from the EDA.

We had three main functions called *validate*, *create_submission* and *create_submission_from_pickled_model*. While we mostly used validation to train our model, *create_submission* was used to predict for the test file and create a submission out of it. We used *create_submission_from_pickled_model* if we had a trained model in pickled form to predict for the test file. The code structure made it very easy to work with different feature sets, different training models and various parameters. Pickled Model

References

[1] Daniele Micci-Barreca. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. 2001.