Two Sigma Connect: Rental Listing Enquiries

Team Name: NVM

Nishant Malpani
IMT2016095
IIIT, Bangalore

Mahidhar Bandaru
IMT2016070
IIIT, Bangalore

Srinivasan Vijayraghavan
IMT2016083
IIIT, Bangalore

DEFINING THE PROBLEM

Two Sigma Connect and RentHop featured rental listing data on Kaggle to predict the number of inquiries a new listing receives based on the listing's creation date, the price of the rent-space and many other features. The motivation to solve such a problem is to help RentHop better handle fraud control, identify potential listing quality issues, and allow owners and agents to better understand renters' needs and preferences.

OUR TAKE ON SOLVING THE PROBLEM

Rather than delving into the problem-solving straight away, we initially laid our emphasis on incorporating naive steps like understanding the problem and the data, brain-storming on what kind of features might be useful, etc. We, then, got into the more conventional steps of data pre-processing, feature extraction and model training. The below list captures the events sequentially which we intend to explore individually in the coming pages.

1) ***Gathering insight from the provided data***
2) ***Exploratory Data Analysis***
3) ***Data pre-processing and transformation***
4) ***Feature engineering***
5) ***Model Selection***
6) ***Hyper-parameters tuning***
7) ***Model evaluation***
8) ***Analysis***

I. GATHERING INSIGHT FROM THE PROVIDED DATA

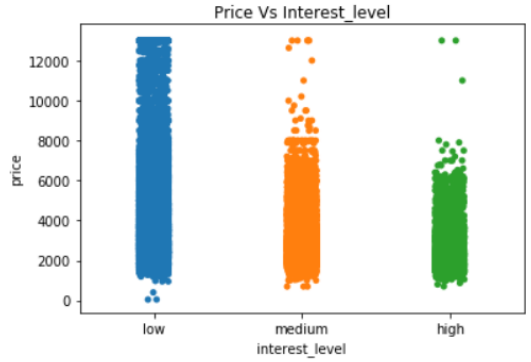
Our team had plenty of spontaneous discussions by placing ourselves in the shoes of the customers; answering questions like what we, as an individual looking for a rental house, would be most attracted to. These brain-storming sessions gave us a huge insight into what kind of a problem we're dealing with and what features can be drawn from the kind of data that was provided.

For example, customers would want their house in a locality which is near to pivotal places like the subway, the airport or maybe even places like parks or public gardens. Customers would want to stay in a particular posh locality although a good deal is always appreciated. People who own a pet might be particularly interested in buildings which allow pets. Manager reputation could also be considered by the people looking for a house. Many other traits like the number of bedrooms, the availability of a bathroom per bedroom, amenities provided in the building (sports room, swimming pools) are what a customer might be looking for in their new house.

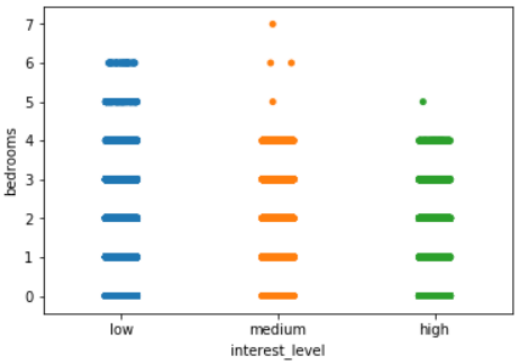
Of course, the answers to such questions are highly subjective and involve various trade-offs but we think it's important to go through this phase to better understand the problem to be solved with the kind of data presented.

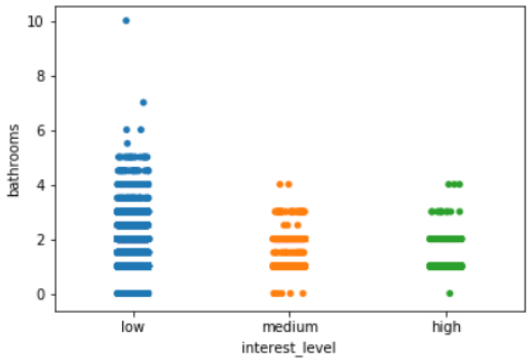
II. EXPLORATORY DATA ANALYSIS

We did spend quite a bit of time trying to understand the data through EDA.
Let us first try to look at how price influences the interest level of a particular listing. As one might expect the most attractive

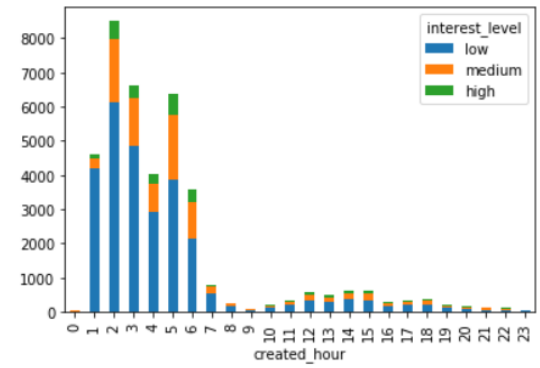


listings generally are the ones which are priced low.
Let's further see the impact of number of bedrooms and bathrooms on the price.

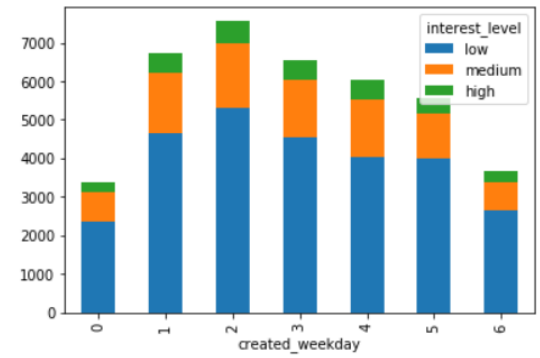




Here we made another plot to see how number of bedrooms affects interest levels. Turns out that people aren't interested in living in houses that are too big. The high interest listings have close to 4 bedrooms at most. Similar conclusions can be drawn for number of bathrooms too.
Let's now look at two plots on date time features.



It looks like most of the listings were posted during the 01:00 and 06:00 hrs. Also the listings posted during 01:00 hrs and 06:00 hrs seem to be more attractive for some reason.



In this plot we can see that the relative ratio of high, medium and low interest listings is more or less same for all days of the week. However, we can see that the number of listings posted on Monday and Sunday are the least.

We did plot more graphs for EDA but for the convenience of the reader we decided to put the jupyter notebook in the project folder.

III. Data pre-processing and transformation

As the raw data provided might contain erroneous figures or missing values, we convert them to a format familiar to the models provided by 'scikit-learn'.

Handling missing values: Although there were no 'NaN' values in the dataset, from the analysis of the dataset we figured that around 7000 buildings (17% of the dataset!) had '0' as their 'building_id'. We handled such buildings by assigning them a 'building_id' of the nearest building, which was calculated using the distance between the buildings given their latitude and longitude. This proved to be an accurate solution for buildings with multiple listings; they all had the same 'building_id', 'longitude' and 'latitude' (distance computed comes close to zero).

Removal of the outliers: The statistics of the dataset suggested columns like 'latitude', 'longitude' and 'price' to have outliers. This was further confirmed by the box-plots made in the EDA. Rather than completely discarding such data points, we handled outliers in the upper range by assigning them values same as the 99th percentile value sampled for that particular column and similarly, the outliers in the lower range were assigned the 1 percentile value.

Feature encoding: Basic label encoding was applied to categorical columns like 'display_address', 'manager_id', 'building_id' and 'street_address' to be able to feed such columns to the models.

Text processing: On analyzing the dataset, particularly text-containing columns like 'features' and 'description', we discovered them to comprise of inessential (in terms of building a robust model) content like HTML tags, symbols other than the alphabets and numbers, etc. So we used regular expressions to set all alphabets to lower-case, remove all symbols other than the alphabets and numbers (HTML tags are removed) and also replaced any number (1-9) to "num" as we didn't want our model to overfit on specific numbers in the 'description'.

Feature scaling and normalization: This step was realized by using *scikit-learn*'s 'StandardScaler', where the features of every column are normalized using the mean and variance of that column to a specific range. Normalization is crucial for simpler models like Logistic Regression, Support Vector Machines, etc.

IV. Feature engineering

Keeping in mind the significance of feature engineering, we dedicated most of our time in this part of the project. To do justice to the amount of effort we've invested in extracting features, we have explained (below) the motivation behind

our features in utmost detail. Also, instead of describing each and every feature independently, we made bundles of related features to offer high levels of readability to the reader.

Price-based features: 'Price' playing one of the most influential roles in our brainstorming sessions, we extracted some basic features like 'price per bedroom', 'price per bathroom' 'price per room' (room = bathroom + bedroom) and 'logarithmic price', as 'price' alone doesn't capture the complete picture; people will be interested in a home with high price provided there are more rooms being offered. The logarithmic function is applied due to the testament of "logarithmic thinking" in humans. Differences in the higher price range don't seem much than in the lower price range, for the same difference.

Time-based features: Features like 'created_month', 'created_day', 'created_hour', 'is_night', 'is_weekend', 'days since posted', etc. were fished out from the time stamp of the listing when it goes online. Our reasoning to use such features is that the view rate during the weekends and the nights are high as people are idle, usually, during those periods. The 'days since posted' feature tells us how old is the listing with respect to the latest listing. The motivation behind this is that people would prefer looking at new listings than the older ones. Note that such features don't necessarily tell us about the interest level directly.

Distance-based features: To gain some more insight, we asked ourselves a simple question: "would we like to rent a house in a more accessible place or in the outskirts of the city?" Of course, one would want their house close from essential destinations like their workplace, subways, airport, hospitals, schools, etc. Ignoring subjective responses, we computed distance (from the latitude and the longitude) of the house from places like "JFK Airport", "Central Park", "Washington Square Park" and "Financial District" (where most popular offices and eateries are located). We also researched safe and most popular localities to live in New York, to find distances from these localities as well as again, people would prefer to live in such places. Also, logically, the most popular localities are often the most accessible ones. A few localities we identified were: "Crotona Avenue", "61st Avenue", "Atlantic Avenue", "Lincoln Avenue" and "J Avenue".

Geographical clusters: Using K-Means Clustering algorithm, we made geographical clusters using latitude and longitude. The number of clusters (=15) was taken such that the total variance of price in each cluster is minimized. The motivation to do so it to group up all alike-priced houses together ergo giving us information on the type of locality the house is based in; people would pick houses, even if they are a bit high priced if it's in a decent locality.

Count and Statistical features: To get an idea of how active a manager is or how popular a building is, we count the number of listings a particular manager has and the number of listings posted in a particular building. We then extended this idea to get the number of listings a specific display and street address has. Moving onto statistical features, we computed the median and the standard deviation of prices when grouped with 'manager_id', 'building_id' and 'cluster_id', individually. Justification of this would be that this gives us an idea about which price range a particular manager deals with or which the type of locality a building is in.

Manager skill and building popularity: We thought that some sort of popularity or skill measure of the manager who posted the listing could give our model useful insights. We generated three features namely manager_level_low, manager_level_medium, manager_level_high each of which gives the ratio of low, medium, high rated listings posted by a manager. We also made the corresponding features for building_level_low, building_level_medium, building_level_high in an analogous way. However, we could not find much of an improvement in the evaluation metric (i.e logloss). Thanks to a kernel we found out on the original kaggle contest we found out that we were facing an issue called data leakage in machine learning. After fixing the leakage our model did wonders using the manager level features. However, the building popularity features didn't show much of an improvement even after fixing the data leakage issue.

Text-based features: We exerted plain TF-IDF Vectorized to fit and transform the 'features' and 'description' column, separately. For the 'features' column, we used ngram_range = (1, 1) only as it doesn't make sense to increase the maximum. In opposition to that, we used ngram_range = (1, 3) for the 'description' column as those are stated as complete sentences.

V. Model Selection

We tried various models and techniques of ensembling (stacking, blending, voting, weighting) for our problem. All the models taught in the tutorial sessions have been tried along with hyperparameter tuning. However, we finally saw that nothing was coming even close to XGBoost in terms of the evaluation metric, i.e logloss. We believe this is because of the strong non linear nature of the problem, which we found out during EDA.

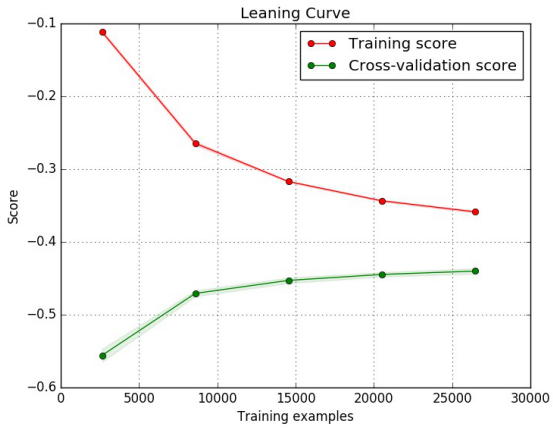
VI. Hyper-parameters tuning

After reaching the limits to our cognitive abilities for further feature extraction, we turned our attention to hyper-parameters tuning to make our model more solid. Instead of blindly employing trail and error method to find the best parameters, we took time to read about the parameters so as to relate with the theory taught in the class and the tutorial sessions. We made use of *scikit-learn*'s RandomizedSearchCV to get an approximate range of values for each parameter and then further used GridSearchCV to get precise values for the same.

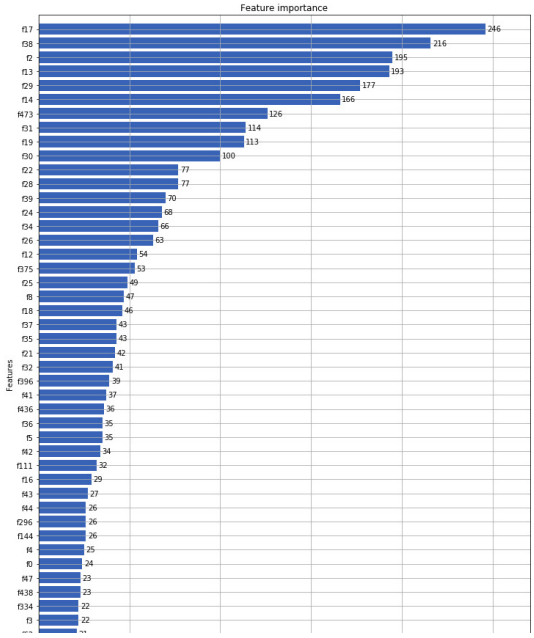
VII. Model evaluation

Evaluating a model's performance is crucial to run through as it tells a lot about how the model will work in the future on unseen data. The "best" model we got is termed the "best" due to comparable evaluation metric (logloss) in both the training dataset and validation dataset, as seen in the learning curve plotted. The plot also gives an idea about where the bias-variance tradeoff lies.

Note: The logloss on train dataset: 0.480554 (for the best model)



VIII. Analysis



We made further deductions on why our best model was in fact better than the others. The main difference we found were the features that we used, which can be further analysed for the 'feature importance' plot attached.

Note: Please go through the drive link for the feature names.

References

[1] https://www.kaggle.com/sudalairajkumar/simple-exploration-notebook-2-connect

[2] https://www.kaggle.com/guoday/cv-statistics-better-parameters-and-explaination

[3] https://www.kaggle.com/den3b81/do-managers-matter-some-insights-on-manager-id

[4] https://www.kaggle.com/c/two-sigma-connect-rental-listing-inquiries/discussion/32358

[5] https://www.kaggle.com/c/two-sigma-connect-rental-listing-inquiries/discussion/32404

[6] https://towardsdatascience.com/fine-tuning-xgboost-in-python-like-a-boss-b4543ed8b1e

[7] https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/