

CMPT 459 Final Report

Group: Information Gainers

By Sina Khalili, Logan Militzer & Phong Le



**Section 1:Exploratory Data Analysis (Authors: Logan & Sina)**

Introduction

When we began to work on the milestone 1 regarding the analysis and exploration of the data, we wanted to become familiar with the data. Becoming familiar with the data would give us a better understanding of it and therefore, our group would be able to work better with it. This included becoming familiar with the attributes and their meanings, the shape of the data, cleanliness of the data and the class.

Features of interest

While we explored the various features, we created various graphs and charts to allow us to better understand the data. These are trends of some attributes that our group determined to be important and interesting. A strategy we used in getting a feel for the data was to create **interactive** graphs. By varying parameters, we could visually get an intuitive understanding of the data. This method is outlined in (Victor, 2011). For example, to decide the cut-off points for the outliers in price we created a plot of all the data with regards to price and varied the percentile to keep – there was a clear difference after a certain point from a right-skewed graph to a proper normally distributed graph. The trends and visualizations of data can be found in the appendix.

*Created*

Figure 1.1 shows us when most listings were created. As you can see, the most popular times are ~1am-7am. We believe these were popular times because people are updating the listing during times of less traffic. Additional reasons may be because of scripts running and updating listings or the listings are being updated or generated from employees outside of New York.

*Price*

Figure 1.2 shows price and which rental price is most popular. You can see that most of the price values are in the $2,000 to $6,000 range. This falls in line with our expectations as New York is an expense place to live and rent in. This is a feature of interest for our group because it can heavily influence the buyer’s decision. This is the case since this is what the renter will be out of pocket at the end of the month.

*Longitude and Latitude*

Figure 1.3 shows longitude and latitude. Our team wanted to confirm the location of the listing to conclude that they are being listed in New York city. As you can see in the histogram, the values for longitude are in the -73 to -74 range and the values for latitude are in the 40 to 41 range. Additionally, we confirmed these longitude and latitude values to be New York via Google (Lat Long Finder). However, while playing with the graphs (as they’re interactive), we discovered a few values that did not fall in the allowable range. These outliers will be further discussed in the data pre-processing section of this report.

**Section 2: Data pre-processing (Authors: Logan & Phong)**

Dealing with missing values & outliers

We first had to ask ourselves “what was considered a missing value and an outlier” with respect to the context of the problem. We all agreed that finding missing values and outliers did not make sense to the non-numerical features created, description, display\_address, street\_address, features, photos, and interest\_level. Features building\_id, listing\_id, and manager\_id were numerical attributes, but they were created from a hashing algorithm and therefore we did not analyze them. Now, we defined missing values as any one of an empty string, null/none value, or a zero entry. The only exception to this rule was for bathrooms and bedrooms, as it is possible for a listing to not contain one of these amenities. Now, with the dataset cleaned of missing values, we look to find and remove outliers.

From figure 2.1, we decided to remove those values not in the 99.5th percentile; in other words, we considered the context of our problem and determined that a price over 1 million is an unreasonable listing and attributed it to mistyped pricing. With figure 2.2, we used Google Maps to check that the latitudes and longitudes map to valid houses (Lars Eilstrup Rasmussen). Once we removed the bottom 0.2% and top 0.3% of values, we noticed that most of the rental listings resided around New York. In terms of analysis, determining the interest\_level class label for rental listings would not benefit from including houses outside of that area. Therefore, we removed all entries not within that –73 to –74 and 40 to 41 range for longitude and latitude respectively from figure 1.3.

Once again, we gave an exception to these outliers for bedrooms and bathrooms. The range of values 0-7 and 0-10 respectively was too small to label any values as extremes, so we left all those housing entries in.

Feature extraction from images and text

For features, we looked at both image and text data. For image features, we looked at the image width and height along with histograms. These histograms looked at image gray scale as well is RGB colours. The idea here was to see if certain colours in the image generated more interest. We also looked at the contours of the image but soon realized this may difficult to train the classifier around given this course is an introduction into data mining.

For text data, we looked for numerous features from the listing description. Once we generated these features, we added them as new columns if we felt they could be important for classification.

One of the features our group looked at was the number of special characters in a listing description. In figure 3.1, you can see most listings had either 0 special characters or around 6 special characters. We found it quite interesting that the second most popular number of special characters was 6 and we were hoping the classifiers would help uncover this phenomenon.

Another interesting feature we discovered via the text description was looking at the most popular words. In figure 3.2, you can see the most popular words. “Elevator”, “Cats Allowed” and “Hardwood Floors” were the 3 most popular words/phrases to show up in the description.

Another text feature we looked at was stop words. Stop words are words such as 'and', 'we', 'my', etc. In this graph, figure 3.3, we can see that most descriptions contain 0 and 20 stop words and this falls in line with our expectations.

Additional features from external datasets

While our group did not pull any features from additional datasets, we will discuss what we would have done. First, we would put ourselves in the renter perspective and ask what generates value to them. The first thing that came to our groups minds was being close to transit, having access to a nearby park and being close other various daily activities. These activities could include being close to a gym, a Starbucks or a Subway for lunch. We would then look for a data set in Kaggle and join the locations of interest to the listings. We would join on the key longitude and latitude. This would signify that the location is close to the listing.

Feature selection

Our group decided to select these features: ‘bathrooms’, ‘bedrooms’, ‘latitude’, ‘longitude’, ‘price’, ‘Special Characters’, ‘Numbers’, ‘StopWord’ & ‘Uppercase’. Of these features, we hypothesized that price would be the feature that contributes the most to interest level of a listing. We thought this would be the case because if a place listing was priced lower than average in New York City, there most likely will be a lot of buzz and talk about the listing. The buzz can be social media shares to friends and family or likes on the listing on a social media page.

**Section 3: Classifiers (Authors: Logan, Sina & Phong)**

Choice of classifiers and libraries

*Logan*

I decided to create the decision tree with guidance from Sina for milestone 2 and decided to create a random forest classifier for milestone 3. For the main library to be utilized, I decided use scikit-learn as their classifiers are well documented and therefore, I can better tune the classifier (Cournapeau). I also used pandas and NumPy to import data, transform it and export it (McKinney) (Oliphant).

For the decision tree and the random forest with no optimizations, the scores were not that great. They can be seen below.

*Decision Tree*

Cross-validation score: ~ 63%

Kaggle log error score: 23.58702

*Random forest*

Cross-validation score: ~ 72%

Kaggle log error score: 6.23692

We were surprised with the results, but we do have to account for the fact that no optimizations were made to either classifier. These are the bare, unoptimized numbers.

For decision tree optimizations, we performed min-max scaling normalization method which gave the attributes similar weights. However, as noted in the lecture slide, this process is sensitive to outliers. Therefore, we ensured to removed outliers during the data pre-processing step. Pruning was also performed on the decision tree. This reduced the number of possible paths or branches a classification could follow. Additionally, we performed an exhaustive search from depth 3 to depth 20. This would allow us to see the most optimized decision tree. This search showed that 4 was the best depth for the decision tree to be at. We also played with the number of folds performed by the cross-validation score. Our group determined that 5 fit best for the data set.

For the random forest, we optimized the scores by generating a min-max normalized data set. This data set was the same one used in milestone 2 and therefore outliers were taken care of. This data set was also proven as the decision tree received a decent score on Kaggle. I also tuned of the parameters of the classifier to further improve this classifier’s score. The parameters that were tuned were n\_estimators, min\_samples, min\_samples\_leaf, max\_features, max\_depth and bootstrap. However, determining these parameters comes at a performance cost.

Below are the scores received after the optimizations.

*Decision Tree*

Cross-validation score: ~ 68%

Kaggle log error score 0.99459

*Random Forest*

Cross-validation score: ~ 74%

Kaggle log error score 2.08911

Ultimately, I think the random forest scores could be further improved with time. I ran out of time as my computer was running hot and becoming slow after trying to determine the best tuning parameters.

*Sina*

It was decided to do an SVM for milestone 2 and after some research, XGBoosting for milestone 3 (Tianqi Chen). As mentioned previously, scikit-learn was used for its breadth of choices and common interface across many classifiers. Numpy, pandas and matplotlib were used in holding, relating, and visualizing the data respectively - this is the recommended Jupyter Notebook stack (Oliphant) (McKinney) (Hunter) (Fernando Pérez, 2015).

The initial results are shown below:

SVM:

CV score ~68%

Kaggle log error score: 1.49068

XGBoosting:

CV score ~71%

Kaggle log error score: 2.02

There is an interesting note here - SVM's took a very small amount of time to understand, but a significant amount time to train. In contrast, the XGBoosting models took a very minimal amount of time to train - at most 5 minutes on the most complex ones, and a matter of a few seconds on the simple cases – however, due to the absolutely massive amount of options and amount of time it took to understand the model, it in fact took longer to work with than the SVM model, while performing worse.

In fact, we noticed that the simpler models usually performed much better without any tuning than the more complex ones - which was an interesting discovery which shows that advanced data mining methods don't necessarily improve all aspects, but instead they become more specialized. Naturally, after a lot of fine-tuning, the advanced models will vastly outperform the older, simpler models. But not without significant work.

To fine tune the models, ample use of the GridSearchCV function was used - which takes a range of hyperparameters and performs cross validation on a model for every possible combination of those hyperparameter values. For the SVM, the value tuned was the 'C' value, gamma value and various kernels (namely radial basis, polynomial and sigmoid). This was enough to produce a suitable SVM. For XGBoosting, the story was different. The absolutely massive amount of options (i.e. learning rate, regulation lambda, regulation alpha, num estimators, etc.) made using the Grid Search unreasonable and instead a Random Search CV was more realistic - which instead of going through every possible combination, it just goes through random combinations for a set amount of time. After a long amount of time spent learning XGBoosting and training it on the data, we achieved a better error - but not by the gains expected by XGBoosting.

The results after tuning were the following:

SVM:

CV score: ~69%

Kaggle log error score: 1.43994

XGBoosting:

CV score: ~80%

Kaggle log error score: 1.22010

Notice that both models were quite difficult to improve. This is another important lesson in data mining: initial gains are impressive and quite large with very minimal amount of effort, and then every incremental gain in accuracy and after takes a substantial amount of effort (spent in tuning hyperparameters, cleaning data, trying new approaches, etc.). Finally of note is that after searching online, we found a particularly well-tuned XGBoosting model for this exact Kaggle which achieved a performance of 0.56 log error score, which is incredible and goes to show what XGBoosting truly is capable of when given the right resources and understanding (Kumar, 2017).

*Phong*

I was responsible for Logistic Regression (LR) for milestone 2 and chose to do K Nearest Neighbours (KNN) classifier in milestone 3 (Navlani, 2018). Another part that influenced my decision is that we covered the topic in class. Scikit-learn was the main library to handle the models and pandas as auxiliary library to clean and visualize the dataset (Cournapeau) (McKinney).

After cleaning the dataset of missing values and outliers, I trained the models with 5-fold cross validation and uploaded my results to Kaggle that yielded the following results:

LR:

CV score: ~69%

Kaggle log error score: 1.43994

KNN:

CV score: ~66%

Kaggle log error score: 11.29661

For LR, I used min-max normalization on the dataset to equally weigh all the values. This was done in order to prevent higher magnitudes from influencing the model more than the lower ones. Also, I went back to review the criteria left on milestone 2. One of the TAs noted that there could be more done to improve LR besides normalizing the values. So, I applied my ideas to optimize KNN to LR; I experimented with the hyperparameters and dimensionality of the feature set. What results I came up with were a feature subset of price**,** bathrooms, bedrooms, numbers, and uppercasewith a C value of 0.10. C is the inverse of the regularization strength. Regularization is a technique to apply penalties to increasing magnitude of values to avoid overfitting for prediction functions. In combination with normalization, it made for some minor improvements.

With KNN, I also used min-max normalization. However, I noted that more features do not necessarily mean a better fitting model. In addition to a risk of overfitting, KNN requires neighbours to be close in every dimension (Grant, 2019). More features add to the dimensionality which increases computational complexity. Using Euclidean distance as the measurement for the nearest neighbours becomes less useful within high dimensional feature spaces. Because of these reasons, LR outperformed KNN in both CV score by 3% and log error score by a factor of ~8. As a countermeasure, I reduced the feature space to just a few variables. In the context of rental listings, I assumed that lower priced houses drew more interest than higher ones, so I chose price as the main variable. The two other features he chose were bedroom and numbers from figure 4.1. Along with this, I set KNN’s hyperparameter n\_neighbors to 10.

The results from the optimizations are as follows:

LR:

CV score: ~70%

Kaggle log error score: 1.04041

KNN:

CV score: ~68%

Kaggle log error score: 1.42207

For KNN, a 2% increase in cross-validation score is a small improvement. The score I want to highlight is the Kaggle log error score. It went from 1.42207 to 11.29661, an almost 8x improvement. Because KNN suffered from the “curse of dimensionality” more than LR, it was important to choose a small subset of features. With LR, I was a little more lenient in the number of features in pursuit of a minimized Kaggle log error score. It went from 1.09819 in milestone 2 to 1.04041 once I gained insight from this report.

Comparison of the results of the different methods

Decision trees are interesting in that they were incredibly poor before optimization and very good afterward showing us that this model is the most susceptible to overfitting. SVM’s on the other hand had the smallest difference between optimization before and after. However, SVM’s started much better with the best pre-optimization score of them all. SVM’s also took the longest to train followed by XGBoosting which took a long time only because it has so many hyperparameters to tune. If time spent learning the underlying algorithm was also a metric, XGBoosting with its complex system, would be first by a landslide. K-Nearest Neighbours had a slight advantage in training time than Logistic Regression. This can be attributed to the fact that KNN is a lazy learner and LR is an eager learner. KNN does not train anything, while LR learns the weights of the coefficients during this time (Raschka). However, as expected, the lack of a training step came at an expense while predicting; KNN took longer to predict interest levels of the rental listings than LR did. This is because every time KNN wants to make a prediction, it searches for the n-nearest neighbours within the entire data set.

**Section 4: Lessons learnt (Authors: Logan, Sina & Phong)**

Most relevant features

To test which features were most relevant, one method used was to hide features, train classifiers on the rest of the features, and then reveal the accuracy to which the data could be handled. Incredibly, if we train a decision tree on only the ‘price’ attribute, we achieved a CV score of 68%. This goes to show the power of the price attribute in handling the prediction accuracy and indeed intuitively price is something that will certainly influence the outcome. Naturally attributes which were highly correlated with price are also considered highly influential (see Fig 4.1) however they are not really giving us any more information in the principal component sense.

Also, according to Fig 4.1, bedrooms and bathrooms had a relatively high correlation with price with respect to the other features. Therefore, using one or the other in combination with price improved the performance of our models. However, these two features are positively correlated. This introduces a problem; increasing complexity of a model by adding an additional feature may not be worth the information learned from it. That is because there is a redundancy of information from highly correlated features. Adding another feature that has no correlation with them can be a countermeasure.

Longitude and latitude, that is, location, was also the next most influential attribute (which did not necessarily correlate exactly with price – though it did slightly). Finally, perhaps disappointingly for rental property owners, the description was of the least amount of influence – at least for the feature we chose to extract (Stop words, numbers, special characters). Intuitively this also makes sense as a description can only change a property’s underlying value (and hence, interest level) so much.

Best classifiers

Our best classifier, in terms Kaggle log score accuracy was, surprisingly, the basic decision tree classifier. This score was 0.99459 - however after milestone 3 we went back and retrained the KNN classifier on only 4 of the features [bedrooms, bathrooms, price, numbers] with min-max normalized data and got an improved score of ~0.82 in Figure 4.2. Also, it is worth mentioning that after viewing another notebook publicly available on Kaggle, we were able to reproduce results of 0.56 with XGBoosting (However, we do not consider this as our own result, and have excluded it from the classifiers section) (Kumar, 2017). After optimizations, the best classifiers are ranked as the following: KNN, Decision Tree, Logistic Regression, XGBoosting, SVM, Random Forest. Worth noting is that the more advanced and newer methods are completely mixed in with the older methods – a result which is surprising given we had assumed that newer methods should be uniformly better. However, with newer methods as mentioned above, hyperparameter tuning and knowledge of when to use which algorithm, in the context of the problem, plays a much larger part in achieving the desired results.

Most efficient classifiers

The most efficient classifier that our group decided on was the decision tree. Why we think it was the most efficient of the classifiers that our group trained was because this classifier only has split at various nodes and create varies combinations. Additionally, the runtime is logarithmic; where m is the number of features. If you want to visualize the decision tree in an image, it is not efficient at that because you need to import a separate library to display it and as we found out, it is computationally intensive.

The least efficient classifier our group determined to be was the support vector machine. The reason we believe it was an inefficient classifier was because it quite simply has a long running time. Additionally, it must look at 50,0002 examples to compute the model.

Overfitting

Overall, overfitting was not a problem. An easy check we did for each model was to look at the scores during cross validation and see if there was a big difference between the training folds and test fold. If the model performed well on the training data and poor on the test, we knew there was overfitting.

We definitely noticed the decision tree was overfitting. This caused the log error score to be over 20 on Kaggle. We remedied this be pruning the decision tree. This then brought the score much closer to 0.

General takeaways

A general takeaway that our group learned was to increase the domain knowledge, even briefly, on the subject we are classifying for. In this case, that would be about rental property listings. This would have allowed us to further focus our intentions to what makes a rental listing property more popular. This knowledge in turn would have led the group to hone in specific features instead of looking at all the features.

Another takeaway, as previously mentioned in the XGBoosting section, was that the more complex models/algorithms produced much worse results if they weren’t tuned – and the same is the case with the contrapositive, that is, a simpler model – say an SVM – will give a better results when it’s not tuned but not improve as much as it gets tuned. This makes sense since it has less degrees of freedom than something like XGBoosting. Furthermore, this also highlights the relative importance of the algorithm choice in the entire data mining process. Before this project we thought that the amount of data and preprocessing, though useful would not be nearly as important as the algorithm choice. However, after the project we realized that indeed data preprocessing and data quality are of the utmost importance and algorithm choice plays a more secondary role – and that’s only after lots of hyperparameter tuning, which we also did not account for! Truly it was an enlightening experience in data mining.

What our group also learned from was to plan ahead for training the classifiers. Why we say this is because some classifiers take a long time to train. We experienced this with the SVM. Our solution was to train it overnight so it would be ready in the morning. This strategy ended up working for us. Another solution would be to use more powerful computers. However, we did not have access to such computing power. We will use this knowledge in the future when creating classifiers for other classes and in our field of work

**Section 5: Recommendations for rental property owners (Authors: Sina & Phong)**

Recommendations to property owners

It is unfortunate, but intuitively (as well as experimentally) true that the points under the control of the property owner (description, tags) are among the less influential attributes for garnering interest. That is, of course, with the exception of price – which is of the absolute utmost importance. A simple and naïve suggestion for generating more interest would be to adjust the price. Mathematically the results show that this will influence the results more than any other attribute. However, of course, this price is usually not something property owners would be willing to change.

As an extension to this, another recommendation for property owners is to undercut prices. Due to this, they should search for other houses with similar features to their own and average the prices. The owner of the rental property would then price their listing lower than an average. This is effective because we saw that pricing alone is a powerful feature to determine the interest level. When potential renters go to a listing website, one of the first features they would select to order the entries by is price.

Listing properties to maximize interest

A good highlight would definitely be the number of bathrooms as those with higher bathroom number seem to be attracting more interest (although this could be seen as a proxy for price). A lot of the features aren’t in the control of the property owner (for example many entries are labelled “pre-war”). In terms of features to highlight, “hardwood floor” and “elevator”. It was not the case that a particular feature was negative so it’s never a bad idea to not put as many features as possible. So, it’s best to put features which are found across the board such as “hardwood floors” “pet friendly” and “elevator” as all high interest listings have these features, as well as have a competitive price.

Properties of a listing to be highlighted

As mention prior, the price of a property will surely affect all aspects of interest and should be adjusted according if at all possible. An in-depth explanation with proper feature tags is preferable to no description at all – though the use of special character (the exclamation point for example) is not really necessary, as it does not improve the chances of becoming a high interest listing. The number of bathrooms is an important factor and the more the better. Having no bathrooms is a great hamper to becoming a high interest post. Finally, proximity to an interesting location, for example coffee shops such as Starbucks and public services like schools or libraries, based on our preliminary research, provided a boost to interest as well. This can also be seen on some of the notebooks on Kaggle (Kumar, 2017).

**Section 6**

**References**

Cournapeau, D. (n.d.). *Scikit Learn.* Retrieved from https://scikit-learn.org/stable/

Fernando Pérez, B. G. (2015, February). Jupyter Notebook. Retrieved from https://jupyter.org/

Grant, P. (2019, July 22). k-Nearest Neighbors and the Curse of Dimensionality. Retrieved from https://towardsdatascience.com/k-nearest-neighbors-and-the-curse-of-dimensionality-e39d10a6105d

Hunter, J. D. (n.d.). *matplot.* Retrieved from https://matplotlib.org/

Khalili, S. (n.d.). For the Two Sigma Connect: Rental Listing Inquiries dataset. Retrieved from https://github.com/SinaKhalili/kaggle-data-analysis

Kumar, S. R. (2017, April). XGB starter in python. India. Retrieved from https://www.kaggle.com/sudalairajkumar/xgb-starter-in-python

Lars Eilstrup Rasmussen, J. E. (n.d.). Google Maps. Sydney. Retrieved from https://www.google.com/maps

Lat Long Finder. (n.d.). *New York City, NY, USA*. Retrieved from https://www.latlong.net/place/new-york-city-ny-usa-1848.html

McKinney, W. (n.d.). *pandas.* Retrieved from https://pandas.pydata.org/

Navlani, A. (2018, August 2). KNN Classification using Scikit-learn. Retrieved from https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

Oliphant, T. (n.d.). *NumPy.* Retrieved from https://numpy.org/

Raschka, S. (n.d.). Why is Nearest Neighbor a Lazy Algorithm? Retrieved from https://sebastianraschka.com/faq/docs/lazy-knn.html

Tianqi Chen, C. G. (n.d.). XGBoost. *A Scalable Tree Boosting System*. Retrieved from https://arxiv.org/pdf/1603.02754.pdf

Victor, B. (2011, October). Up and Down the Ladder of Abstraction. Retrieved from http://worrydream.com/LadderOfAbstraction/

Waskom, M. (n.d.). *Seaborn.* Retrieved from https://seaborn.pydata.org/