SPRING 2020
CMPT459
PROJECT Final Report

Zhilin Lu (301269999)
Kainoa Seaman (301351391)
Daniel Lee (301318659)

Repository
https://github.com/danlee0528/CMPT459-DataMiningProject

Dataset
https://www.kaggle.com/c/two-sigma-connect-rental-listing-inquiries/overview

p.s: we didn't want to reduce the content by having a middle margin, our format is still 1 inch margin from both sides

Exploratory Data Analysis (Dainel Lee)

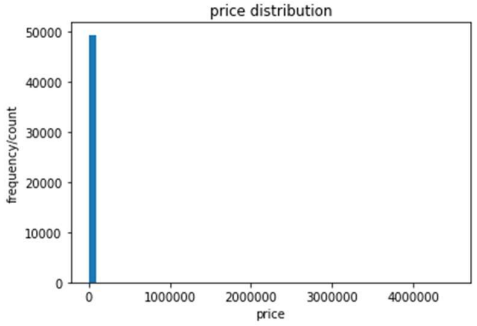
In statistics, exploratory data analysis (EDA) analyzes data sets to summarize their main characteristics, often with visual methods. It is intended to seek what the data can tell us beyond hypothesis testing tasks and possibly lead to new data collection and experiments. In this project, we chose two graphical techniques to visualize the data: box plot and histogram

A box plot depicts groups of numerical data through their quantiles and have lines extending from the boxes or whiskers indicating variability outside the upper and lower quantiles. It can detect outliers as individual points and display variation in samples of statistical population without making any assumptions of the underlying statistical distribution.

A histogram is an approximate representation of the distribution of numerical or categorical data. It is constructed by dividing the entire range of values into a series of intervals and counting how many values fall into each interval. Histograms give a rough sense of the density of the underlying distribution of the data and often allow density estimation.

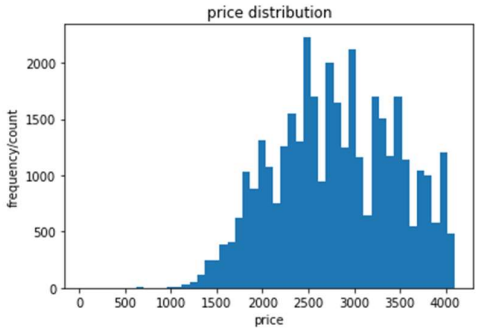
Distributions for the most important attributes

For this project, we were encouraged to discover the proportion of values of numeric columns such as Price, Latitude and Longitude. These columns were found to have 49352 entries with various ranges of values. As our primary purpose of EDA was to visualize the initial data, we considered exemptions of the biggest value of each numeric column from histograms if it dominated histograms for the sake of visualization. Otherwise, it would cause histograms to look skewed. For example, the initial histogram of Price column didn't reasonably visualize the proportion of its values as below:



<Histogram of Price without Filtering>

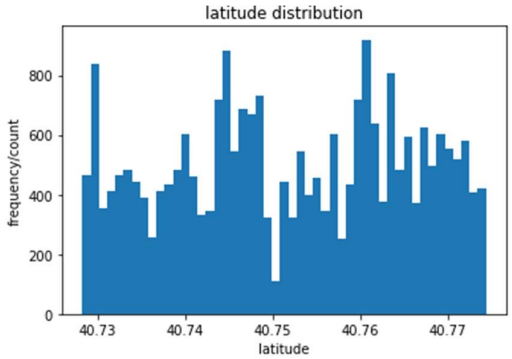
Therefore, it was necessary for us to take a certain percentage of entries until we could vividly see it. When we took 75% quantile of Price column, the histogram produced more meaningful results follows:



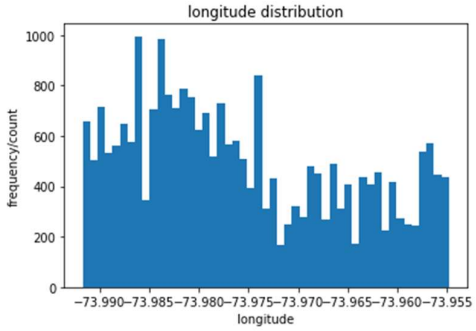
<Histogram of Price within 75% quantile>

With the first histogram, it was found that the mean price of 49352 listings was approximately 3830 dollars and could be as high as 44900 dollars or as low as 43 dollars in USD. The later histogram had the mean price of 2808 dollars for 37097 listings, and it could be as high as 4100 dollars or as low as 43 dollars (in USD).

Similarly, Longitude and Latitude columns without filtering showed that the mean values of 49352 listings were -73.955716 and 40.741545, each with max values being 0 and 44.8835. With proper adjustment of quantile, Longitude had the mean of -73.975782 with the max of -73.95748 and min of -73.9917 for 75% quantile, and Latitude had the mean of 40.751827 with the max of 40.7743 and min of 40.728300 for 75% quantile.



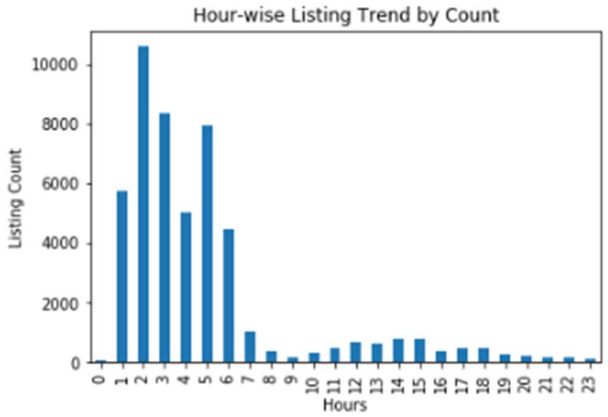
<Histogram of Latitude within 75% quantile>



<Histogram of Longitude within 75% quantile>

Listing trends over time

To explore the listing trends over time, we compared the number of listings created by hour and interest level as follows:



<Hourly Listing Trend>

From the histogram, we were able to identify the top five busiest hours of rental listings which were 2 PM with 10596 listings, 3 PM with 8318 listings, 5 PM with 7954 listings, 1 PM with 5749 listings and 4 PM with 5021 listings.

Data Pre-processing

Dealing with missing values, outliers (Kainoa Seaman)

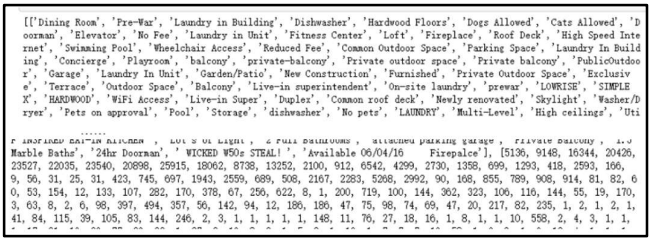
Dealing with missing values depends on the attribute, which determines whether we can drop the value, or replace it with a default value. For example, bathrooms and bedroom counts will have default values of zero if data is missing. But in some cases these zero values are not considered missing data. For studio apartments, the living room is considered the sleeping space, and will correctly hold a value of zero. For all listings however, there must always be a bathroom. With this information, we can infer that listings with 0 bedrooms but >= 1 bathroom is correct data, but listings with 0 bedrooms and bathrooms is incorrect, and data must be removed.

Instances with missing description fields can be kept, because listings with no description could help determine interest. Instances with missing _id values values should be kept, as _ids won't influence interest.

Feature extraction from images and text

Text Extraction (Zhilin Lu)

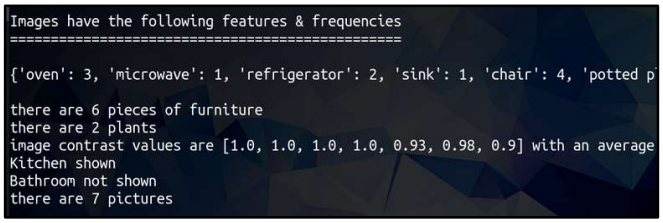
The 'features' attribute is being chosen for text extraction, which the reason will be explained later in the Feature Selection section for Decision Tree and Bernoulli Naive Bayes Classifier. It will tokenize each phrase in each record of 'features', then it will also count and record the times of appearance of each phrase. So eventually, it will generate a 2-D array that contains each phrase itself and the appearance times of the corresponding phrase



<2-D array contains common word and corresponding appearance times>

Images Extraction (Kainoa Seaman)

The cvObjParser.py file (found in our repo) takes two arguments, one for image output, and the other for the folder containing all the images for a listing. At a high level, we believed a staged listing (one with furniture) will make a listing feel more 'homely' and increase interest, along with having plants, more images, and high contrast values. We infer high contrast means images have been post-processed, and will be more appealing. We also infer having more images than less images will create more interest. The script uses a pre-compiled computer vision model from a python library (see reference section). The script will find furniture, appliances, and plants. The script will also extract contrast values, calculating the contrast average and count the number of images.



We never used this function as a feature however, since it required scraping all images from each listing across the database, and Kaggle only provided a subset of sample images. If we had all of the images on the website, we could have implemented this feature.

Feature selection

Decision Tree and Bernoulli Naive Bayes Classifier (Zhilin Lu)

For both Decision Tree Classifier and Bernoulli Naive Bayes Classifier, the features that are selected are 'bathrooms', 'bedrooms', 'price', 'features', 'created'. The reasons are, firstly, all id-related features are removed because they are unique for each record, which has no effect on final result prediction. Secondly, we choose 'features' among 'features' and 'description' because text in 'features' are mostly phrases, and 'description' has long sentences that have too much unrelated text. Thirdly, 'longitude', 'latitude', 'display_address' and 'street_address' have lower correlations to 'interest_level' than 'bedrooms' and 'bathrooms'. So we also choose to remove those features.

However, there are differences on feature transformation for Decision Tree Classifier and Bernoulli Naive Bayes Classifier. For Decision Tree, values in 'features' and 'created' have been transferred to numerical values, which are called 'common word count' and 'create time interval'. And for Bernoulli Naive Bayes Classifier, besides those changes in Decision Tree Classifier, there is extra changes for the 'price' feature, which it is being transferred into the 'price_interval' feature and uses

numbers to represent categorical feature, so it could provide discrete value for Bernoulli Naive Bayes Classifier.

SVM (Dainel Lee)

SVM for multi-class classification problems has limited choice of features. In this project, there were three class labels that we wanted to predict from the dataset - low, medium and high interest levels. The number of features used to predict these labels must match the number of those labels according to the implementation requirement of SVM. Because of this constraint, I used Heatmap to find the top 3 most correlated features to our target variables - bathrooms, bedrooms and price.

KNN (Dainel Lee)

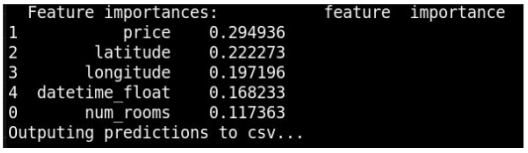
KNN was more flexible than SVM in terms of selecting features therefore we were able to try various methods to select features. First of all, feature importance method selected datetime, price and number of photos as the top 3 most important features. Feature importance selected these features based on scores assigned to each of them. The higher the score the more important or relevant is the feature towards the output variable. Secondly, univariance selection chose total number of rooms, bathrooms and bathrooms as the top 3 strongest relationships with the target variable among non-negative features. Lastly, heatmap chose bathrooms, bedrooms, and price as the most correlated features to the target variable. Among these options,we decided to take the result produced by the feature importance algorithm. The reason is that Chi-square only deals with non-negative variables and it doesn't take negative variables into account such as latitude and longitude.Similarly, the heatmap only shows the correlation between features, not with respect to the output variable we seek. Therefore, the top 6 attributes returned by the Feature importance algorithm were selected: price, datetime, latitude, longitude, number of photos and number of text features.

Logistic Regression (Kainoa Seaman)

The logistic regression algorithm does not have a parameter for choosing the most important features, so we conducted our own analysis. For feature selection, we chose bathrooms, bedrooms, latitude, longitude, price, number of text features, number of photos, and hour of submission. Each feature seemed to bring some relevance to interest from a high level. More bathrooms and bedrooms would be more interesting to most, latitude/long could determine a city that is popular or dense. Number of amenities, and photos would improve interest, and price is the biggest determination of interest for potential renters. In modification of the logistic regression classifier, I reduced features to bathrooms, bedrooms, and price.

Random Forests (Kainoa Seaman)

For feature selection I used a number of rooms, latitude, longitude, price, and datetime_float, which is a timestamp of each listing. The model seemed to infer from the timestamp the important hours, minutes and such that listings would be more important. In random forests, a subsample of features are chosen for each tree in the forest, and the model would choose the most relevant features. This allowed for some leeway with choosing features. The model found price to be the most important feature, followed by lat, lon, datetime_float, and number of rooms, which seemed to show minimal relevance.



The features selected with random forests contrasts what we found in the logistic regression model, where we thought price, bedrooms, and bathrooms were most relevant.

Classifiers

Choice of classifiers and libraries

Decision Tree & Bernoulli Naive Bayes Classifier (Zhilin Lu)

First classifier being used is Decision Tree Classifier, which uses following libraries: DecisionTreeClassifier from sklearn.tree for modeling, train_test_split & StratifiedKFold for cross-validation, GridSearchCV from sklearn.model_selection for finding best parameters of Decision Tree, and plotting tools such as plot_confusion_matrix, and the log loss calculation as log_loss from sklearn.metrics.

Second classifier being used is Bernoulli Naive Bayes Classifier, which uses following libraries: BernoulliNB from sklearn.naive_bayes for modeling, train_test_split & StratifiedKFold for cross-validation, and plotting tools such as plot_confusion_matrix, and the log loss calculation as log_loss from sklearn.metrics.

SVM (Dainel Lee)

SVM is an inherently binary classifier that tries to separate two classes with a hyperplane. When it is used with multi-class, it needs a correct set of parameters to achieve the best classification results for any given problem. Parameters that may result in an excellent classification accuracy for problem A, may result in a poor classification accuracy for problem B. Therefore, experimenting with different parameter settings to achieve a satisfactory result Is mandatory. In other words, it is prone to overfitting if the number of features is much greater than the number of samples. To design SVM, Sklearn, Numpy, Pandas and other miscellaneous libraries were used.

KNN (Dainel Lee)

KNN is capable of finding hyperplanes or best decision boundaries to classify data. However, KNN is a lazy learning classifier that requires high computation power and large memory to store each data point it classifies. It was implemented with Sklearn, Numpy, Pandas and other miscellaneous libraries.

Logistic Regression & Random Forests (Kainoa Seaman)

I used linear_model.LogisticRegression, kfold, and stratifiedkfold python libraries for logistic regression. I used ensemble.randomForestClassifier, gridSearchCV, and RandomizedSearchCV for random forests, and used numpy and pandas, linear_model, and confusion matrix for both classifiers.

The RandomForestClassifier algorithm was used on milestone 3 to see if ensemble algorithms could produce better performance scores than logistic regression.

Optimization of classifiers

Decision Tree and Bernoulli Naive Bayes Classifier (Zhilin Lu)

For the optimization of Decision Tree Classifier, there are three major improvements have been taken. First action is improving Cross-Validation by using Stratified K-fold (5-fold). This will divide the dataset into 5 sets, including 4 training datasets and 1 validation dataset. Each set contains approximately the same percentage of samples of each target class as the complete set. Second action is Pre-Pruning by using Grid Search. It will use a range of parameters provided to find the certain parameter that could build the model with the highest accuracy score. Third action is Post-Pruning by using Minimal Cost-Complexity Pruning. It uses the 'ccp-alphas' attribute of the best classifier that is found through Pre-Pruning. It then chooses the model with ccp-alpha that achieves the best accuracy score on the validation dataset.

For the optimization of Bernoulli Naive Bayes Classifier, there are two major improvements have been taken. First action is feature selection. Because the 'common word count' feature still has too much information, it is being transformed into several binary features, which the number of features equals to the number of common words being used in the dataframe. Then each binary feature represents one common word. And if the value of one binary feature 'X' equals 1, that means in 'feature' of this record, this common word 'X' has appeared at least once. In this method, it will be much easier to see the influence of each common word on the label. Then the feature selection is applied to dataframe, which is L1-based feature selection. It will select features that have non-zero estimated coefficients. After this, the number of features in dataframe will be around 5 to 10. Second action is improving Cross-Validation by using Stratified K-fold (5-fold). This will divide the dataset into 5 sets, including 4 training datasets and 1 validation dataset. Each set contains approximately the same percentage of samples of each target class as the complete set.

SVM (Dainel Lee)

One of the drawbacks of SVM is that it is liable to overfitting. To identify any overfitting occurred during my training, I compared the accuracy rate of two SVM classifiers: a default SVM without 5-Fold Cross-Validation and another SVM modified with 5-Fold Cross-validation. Because the 5-Fold Cross-Validation generated a variety of trainsets and testsets for the SVM model, I could know when the accuracy rate of the default SVM was higher than the modified SVM and was over 0.5, then there was overfitting. Hence I ended up with two accuracy rates that were close and exceeded 0.5 against the testset (test.json), I didn't have high variance or overfitting towards the given testset. However, the Kaggle produced very high variance in terms of scores. Each score was really off from what I had when training the SVM model on the given testset.

To improve the classifier, I experimented with the value of its parameters. SVM has the 'gamma' parameter which defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. It is a parameter for non-linear hyperplanes and SVM tries to exactly fit the training data set with higher gamma values. Decreasing the gamma parameter reduced overfitting given the same gamma value and K-fold CV, however it was not enough. So, I changed the '*C*' parameter which trades off misclassification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly by giving the model freedom to select more samples as support vectors. Decreasing "C" parameter (less penalty) reduced overfitting better given the same C value and K-fold CV. Nevertheless, the 'K' in the K-Fold Cross-Validation which refers to the number of groups that a given data sample is to be split into could be improved. The higher the K, the more groups of datasets we have. But it should not be too high or too low, so that it can generally result in a less bias. When increasing the value of the K parameter from the K-Fold Cross-Validation, it also helped mitigate overfitting slightly. Lastly, I changed the type of K-Fold to Stratified K-Fold which returns unique sets that contain approximately the same percentage of samples of each target class as the complete set. Stratifying the K-Fold help mitigated overfitting given the same C value and gamma value.

KNN (Dainel Lee)

For KNN classification, the output is a class membership and an object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbours. It is a type of lazy learning where the function is only

approximated locally and all computation is deferred until function evaluation. The best choice of k depends upon the data, generally, larger values of k can reduce the effect of the noise on the classification, but they make boundaries between classes less distinct.

There were several techniques attempted to improve the classifier in this project. For KNN, finding the value of k is crucial. A small value of k means that noise will have a higher influence on the result and a large value makes it computationally expensive. Data scientists usually choose an odd number if the number of classes is 2 or select k that is sqrt(n) and I decided to use sqrt(n) as the maximum threshold of the KNN model. KNN also has different algorithms for finding nearest neighbours. BallTree is a space partitioning data structure for organizing points in a multi-dimensional space. It creates partitions of data points into a nested set of hyperspheres known as "balls". The resulting data structure has characteristics that make it useful for nearest neighbour search. KD-tree is another space-partitioning data structure for organizing points in a k-dimensional space. My initial choice of features didn't produce much difference even though I tried different ranges of K value and searching algorithm. So, I tried different sets of attributes given by the Feature Importance algorithm. Furthermore, stratified K-Fold and increasing the value of K were attempted subsequently. Stratifying the K-Fold reduced overfitting quite a bit, however it failed to make a distinct difference from changing the parameter setting of KNN classifier.

Logistic Regression (Kainoa Seaman)

For cross-validation we used a k-fold algorithm. kfold divides all samples into k-1 groups of samples of equal size, and one test set. I have tried 10-fold, and stratified 10-fold, which better samples the minority classes, but they did not increase accuracy in my case. I did not count cross validation as an optimization however, the actual optimizations are as follows:

First optimization was to reduce the number of features. Some of these features might have dependence on other features, or not help to increase model accuracy. I removed text and image features created from m1, and left bathrooms, bedrooms, latitude, longitude, price, and hour of post.

For the next optimization, I tuned parameters of the logistic regression algorithm. I set multiclass to multinomial which can categorize labels other than binary (as logistic regression is typically for binary labels), and is at default set to ovr which stands for one over all. If the logistic regression algorithm finds more than 2 labels, ovr would convert each label to binary and test the probability of that label being true vs all other label likelihood. I changed the solver algorithm to saga, which is set to lgfgs at default and typically works better for smaller datasets. saga is meant to work better for larger datasets. Like ours.

For the last optimization, I turned on warm_start, which reuses solutions to previous fit calls, which seemed to work well with our k-fold cross validation. I set class_weight to balanced, which adjusts label weights depending on class frequencies. I noticed that interest levels of low, medium, and high were not evenly distributed, so this setting made sense.

Random Forests (Kainoa Seaman)

To perform cross-validation, I used a cross validation parameter within the GridSearchCV algorithm. The gridsearchCV algorithm allowed me to choose a range of hyperparameters to tune while also allowing for k-fold cross validation. I used a 5-fold cv. Cross validation is not needed with random forests, since it is an ensemble algorithm, but I used it anyways, and decreased the number of trees that were created to

drastically reduce the run-time of the random forest algorithm. Apart from cross validation, the optimizations are as follows:

The First optimization was using the random hyperparameter grid algorithm. The algorithm took in a large range of parameters to be used in the random forest algorithm. using the parameter ranges, the algorithm would randomly choose sets of parameters to use in the ranges iteratively, for x amount of loops. The random grid algorithm would then output the best parameters to be used on our dataset. This algorithm would run for about 45 minutes to an hour on my computer, so I would only want to use this algorithm to find a rough estimate as to which parameter values to use for the random forest.

The next optimization was called 'grid search with cross validation'. This algorithm is similar to the last, but instead of taking random samples of random forest parameters, it would exhaustively search every combination of parameters. Taking what was learned from the random hyperparameter algorithm, I used the best parameters chosen from that algorithm, and input them into the grid search algorithm using a smaller range above and below each parameter. This algorithm took between 10 and 15 minutes.

Last optimization I decided to use a few less features. I removed the number_of_photos and number_of_text_features... features, and combined the bathrooms and bedrooms features into a number_of_room features.

Accuracy on cross-validation and Kaggle

Decision Tree and Bernoulli Naive Bayes Classifier (Zhilin Lu)

For the accuracy of classifier in cross-validation, the Decision Tree has the accuracy of 69.217% on training dataset, meanwhile the accuracy for validation dataset is 68.625%.

For the accuracy of the classifier in cross-validation, the Bernoulli Naive Bayes Classifier has 67.341% of accuracy, meanwhile the accuracy for validation data is 67.361%.

For the accuracy of on Kaggle, the score of Decision Tree is 1.126. And the score of Bernoulli Naive Bayes Classifier is 1.07115.

SVM (Dainel Lee)

The first version of SVM classifier achieved about 67% of accuracy on the cross-validation dataset and yielded 1.2688 Kaggle score. After modifications, it achieved 67.47% of accuracy on the cross-validation dataset and yielded 0.822296 Kaggle score.

KNN (Dainel Lee)

The first version of KNN classifier achieved about 88.59% of accuracy on the cross-validation dataset and yielded 2.62211 Kaggle score. After modifications, it achieved about 67.7% of accuracy on the cross-validation dataset and yielded 1.38723 Kaggle score.

Logistic Regression (Kainoa Seaman)

The first version of the classifier received a performance score of 68.12% on validation set, and 2.19 on the test dataset using the kaggle scoring system. The first version also included cross-validation. The performance scores were incredibly high, but this was to be expected as no parameters were tuned before training the model.

Performance on the validation dataset did not change much with optimizations. All of the scores hovered around 68% accuracy. The kaggle scores decrease with modifications however, and lower being better. Starting with the original score of 2.19, reducing some features brought the score down to 1.44. Next I changed the

solver algorithm, and multiclass parameters to achieve a score of 1.35-1.36. The last modification of warm start and balanced weights brought the kaggle score down to 1.09.

Random Forests (Kainoa Seaman)

The first version of the random forest model received a performance score of 99.9% on validation set, and 2.98 on the test dataset using the kaggle scoring system. The scores were found using cross validation. The validation score was very high and implied a high bias was occuring, as is the case with decision trees and random forests. The test score was also high which implies high variance. These scores are not good, and more accurate scores were found with optimization.

With optimizations, performance on the validation dataset actually decreased from 99.9% to 78%. From what we know about overfitting (and explained in more detail later), we know that 78% is actually more accurate to the real validation score, as 99.9% shows overfitting. Kaggle scores did decrease to about 1.61. With the random parameter algorithm, a kaggle score of 1.78 was created, 1.68 with grid_search, and 1.61 with reduction of a few features.

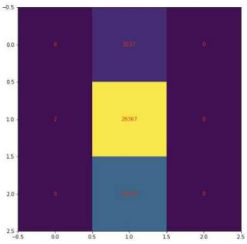
Comparison of the results of the different methods

Comparison of Decision Tree and Bernoulli Naive Bayes Classifier (Zhilin Lu)

Comparing Bernoulli Naive Bayes classifier with the Decision Tree classifier, there is slight improvement on Kaggle website score. The accuracy score on Kaggle website improves from 1.126 to 1.07115. The reason behind this improvement is for Bernoulli Naive Bayes classifier, the right type of features are used, for example, Bernoulli Naive Bayes classifier needs discrete data for features, and it especially needs binary features. After transfering features in the original dataframe into binary features, such 'price' feature being transferred to 'price_interval' feature, there is a slight improvement occurred. Also there is also a second reason for accuracy improvement, which is to expand the 'common word count' feature to several binary features. In the 'common word count' feature, it contains the number of common words that appear on this record. But there is too much information in it, and it's really hard to determine how each common word affects the final predicted result. So after transferring each common word into one feature, it will be easier for the model to determine how each word influences the final predicted result, which could improve the accuracy of the model.

Comparison of Logistic Regression & Random Forests Author: Kainoa Seaman

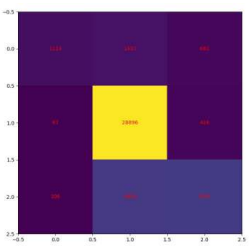
I will be comparing the random forest classifier to the logistic regression classifier. Performance score on the validation dataset for logistic regression was 68% and random forests had a score of 78%. While random forests had higher validation performance scores, the kaggle score was not great. Logistic regression had a kaggle score of 1.09, and random forests had a score of 1.61. It could be inferred that while random forests had greater accuracy in classification, the variance was still quite high.



If we compare the two confusion matrices (logistic regression top and random forests below), random forests seemed to have less bias. We see in logistic regression, there are almost no entries in the low and high columns, however random forests have entries across the diagonals. Bias is still noticeable with random forests, however much less. Finally with log loss analysis, linear regression had 1.014 log loss, and random forests had 0.53 log loss. Since log loss is a measurement of

entropy in our models, with lower scores being better, random forests had less entropy.

SVM and KNN (Dainel Lee)



Comparing KNN classifier to the SVM model, the main difference between these models is that KNN is flexible in terms of feature selection. The SVM model was inherently a binary classifier that tries to separate two classes with a hyperplane. When being used for multi-class problems it needed a correct set of parameters to achieve the best classification results for any given problem. For instance, to predict 3 multi-class labels, I needed to select a set of 3 attributes and various phases of training. However, KNN can have as many or few features as needed regardless of the number of prediction class labels. KNN is a supervised multi-class classifier, in other words, it doesn't need a complex phase of training like SVM to find the best fit hyperplane. The main gains of the KNN model in comparison to SVM are that feature selection plays a big role in determining the prediction accuracy of a model and it is easy to configure for multi-class classification problems. However, KNN is as much computationally expensive as SVM as it searches the nearest neighbours for the new point at the prediction stage and requires high memory to store all the data points. Otherwise, it will be sensitive to outliers which impact its accuracy.

When it comes to performance, SVM achieved 67% accuracy rate on the cross-validation set, 1.07093 Kaggle score. In addition to these metrics, SVM was found to have 1.008 multi-class logarithmic loss. On the other hand, KNN achieved 67.7% accuracy rate on the cross-validation set, 1.38723 Kaggle score and 0.80317 multi-class logarithmic loss. From these results, SVM seemed to produce more reliable predictions on both Kaggle and cross-validation set regardless of information loss which is represented by the value of the logarithmic loss.

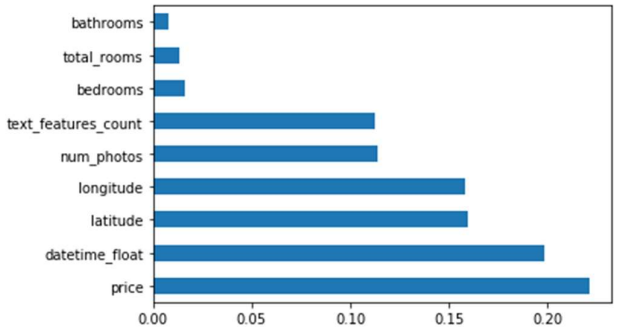
Lessons Learnt (All group members)

Which features were most relevant and why?

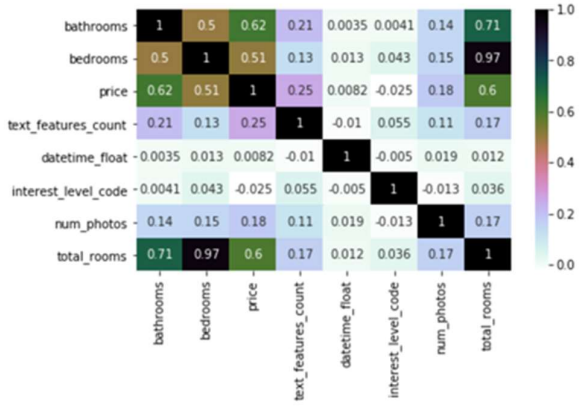
Each classifier trained in the project yielded the best results with the following sets of features:

Modified Classifier	**Features Selected**
SVM	price, bedrooms, bathrooms
KNN	price, datetime, total_rooms, long
Logistic Regression	price, hour, bed, bath, lat, long
Random Forests	price, datetime, total_rooms, lat, long
Decision Trees	bathrooms, bedrooms, price, common word count, created_time_interval
Bernoulli Naive Bayes Classifier	bathrooms, bedrooms, price_interval, created_time_interval, common_word_1,..., common_word_20

From the table above, we were able to find a set of most relevant features that affects the interest level of a rental listing significantly - price, bedrooms, bathrooms, location (longitude and latitude) and word count. Although price, bedrooms and bathrooms were the most common and intuitive features, we can't generalize that only these features are the most relevant. For example, the feature importance method chose price, datetime, location and the number of photos as the most important features.



The heatmap, however, chose bathrooms, bedrooms, price, text_features_count, number of photos as the most correlated features.



Based on these results, we were able to generalize that price, bedrooms and bathrooms are the most common and relevant features that determine the interest level of a listing.

Which classifiers worked best and why?

The score of each classifier is shown in the table below.

Classfier	**TestSet Score**	**Kaggle Score**
SVM	0.6747	0.82293
KNN	0.677	1.38723
Decision Tree	0.69217	1.126
Bernoulli Naive Bayes Classifier	0.67341	1.07115
Logistic Regression Classifier	0.68	1.09
Random Forest Classifier	0.78	1.61

From the scores in the table above, we found that SVM was by far the most accurate classifier in our analysis on the rental listing dataset. The reason is that SVM has a regularisation parameter that allows us to avoid overfitting and solve complex problems with the kernel trick. There are more advantages of SVM over other classification models in this project. However, that's beyond the scope of this introductory data mining course. On the other hand, SVM classifier has trade-offs between accuracy and training time. Storing the kernel matrix requires memory that scales quadratically with the number of data points. Training time for traditional SVM algorithms scales super-linearly with the number of data points. So, these algorithms aren't feasible for large data sets.

Which classifiers were more efficient to train?

KNN and SVM were not efficient to train as they required high computation power and large memory which made them very time consuming.

Decision trees were also not efficient as pruning could be very time consuming. Pruning computation was

based on the tree size and parameters that are chosen by pruning function.

Bernoulli Naive Bayes Classifier is a candidate for the most efficient classifier, which takes about one minute for computing and provides a good Kaggle score.

Similar to Decision trees, random forests were not efficient. Tuning the hyperparameters to find the best accuracy was computationally demanding.

Logistic Regression was another candidate for most efficient classifier, the trade-off being the accuracy, as other classifiers were more accurate.

Bernoulli Naive Bayes and Logistic Regression classifiers were the most efficient classifiers found in our analysis.

Was overfitting a problem? If so, how did you address it?

Each classifier had specific overfitting issues, so we will describe issues for each classifier:

When the Decision Tree Classifier is being used, the overfitting could significantly influence the predicted result. The accuracy score of the first Decision Tree without pruning is as high as 27.13, which is basically predicting the wrong result for every record in test.json. Then after pruning, the accuracy score is being improved to 1.126, which is an acceptable score. Second question is how to address overfitting, which could be achieved by observing the accuracy score of the model on training dataset and validation dataset from cross-validation. The ideal result is that these two scores should approximately equal, which means no significant overfitting occuring. If the accuracy score of the training dataset is much higher than the score of the validation dataset, that means there is an overfitting and it has a neglected impact.

There is a very high likelihood that overfitting will occur in random forests. Having an original score of 99.9% on the validation set is proof that overfitting did occur in the random forest model. An easy solution to reducing overfitting is using more trees in the random forest. increasing the n_estimators parameter allowed for more trees to be created, and like other ensemble algorithms, the best trees would be used to reduce overfitting.

Overfitting in logistic regression classifiers is avoided with the regularization parameter, and is enabled by default as a means to reduce overfitting. I tried to remove regularization to see how much my score decreased, but found its removal is only available in earlier versions of python logistic regression. There is a parameter called C, to adjust the strength of regularization. I tried tuning it to check for overfitting, but not much seemed to be detected.

Overfitting in SVM and KNN is very common.To avoid or minimize overfitting in SVM, the 'gamma' parameter which defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close', was set low. However it was not enough to avoid high variance. Thereby, the 'C' parameter which trades off misclassification of training examples against simplicity of the decision surface was set low. Additionally, when increasing the value of the K parameter from the K-Fold Cross-Validation, it helped mitigate overfitting slightly. Lastly, I changed the type of K-Fold to Stratified K-Fold which returns unique sets that contain approximately the same percentage of samples of each target class as the complete set.

To address overfitting in KNN, I decided to use sqrt(n) as the maximum threshold of the KNN model based on the common practice of data scientists. KNN also has different algorithms for finding nearest neighbours. BallTree is a space partitioning data structure for organizing points in a multi-dimensional space. It creates partitions of data points into a nested set of

hyperspheres known as "balls". The resulting data structure has characteristics that make it useful for nearest neighbour search. KD-tree is another space-partitioning data structure for organizing points in a k-dimensional space. My initial choice of features didn't produce much difference even though I tried different ranges of K value and searching algorithm. So, I tried different sets of attributes given by the Feature Importance algorithm. Furthermore, stratified K-Fold and increasing the value of K were attempted subsequently. Stratifying the K-Fold reduced overfitting quite a bit, however it failed to make a distinct difference from changing the parameter setting of KNN classifier.

Best Chosen Classifier: SVM

We determined accuracy to hold a higher weight in contrast to efficiency when choosing the best classifier for this dataset. Efficiency matters when training the data, but once it is trained, it is no longer computationally heavy. SVM had a significantly higher accuracy score than the rest of the classifiers, and for that reason we determine it to be the best.

Recommendations for Rental Property Owners

Parameters for maximizing Potential Renter Interest for New Rental Properties

Accordingly to our findings, we recommend rental property owners to:

1. Set a rental price to be lower than the average price of 3830 dollars in USD.
2. Plan your listing ahead of the busiest hours which are between 1pm and 5pm
3. The rental price, number of bathrooms and bedrooms are the most important information to potential tenants.
4. Location is also a small factor to potential tenants, but it is not more important than the rental price, number of rooms.
5. The Pet-Allowed apartment usually has a higher interest level, since 'Dogs allowed' and 'Cat allowed' have the most significant influence among 20 of the most common words.

Highlighted Parameters for promoting Existing Rental Properties

For promoting rental properties, we recommend rental property owner to:

1. Adjust the existing rental price to be lower than the average of 3830 dollars in USD.
2. When promoting any existing rental properties, plan ahead of the busiest listing hours which are between 1pm and 5pm
3. The number of words used in descriptions is also a small factor to increase the level of interest of your listings - the more words there are the more likely tenants be interested
4. Change floor type to hardwood floor, which could raise the interest level.
5. Add a Dishwasher for the apartment if there was none, which has a high probability for obtaining a high interest level.

References

1. Random Forests Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

2. RandomizedSearchCV Documentation: https://towardsdatascience.com/optimizing-hyperparameters-in-random-forest-classification-ec7741f9d3f6

3. An Implementation and Explanation of the Random Forest in Python: https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76 Optimizing

4. Hyperparameters in Random Forest Classification: https://towardsdatascience.com/optimizing-hyperparameters-in-random-forest-classification-ec7741f9d3f6

5. Logistic Regression Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

6. Logistic Regression in Python: https://realpython.com/logistic-regression-python/

7. Confusion Matrix Documentation: https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-download-auto-examples-model-selection-plot-confusion-matrix-py

8. Computer vision library for Furniture Recognition: https://github.com/arunponnusamy/cvlib

9. Decision Tree Documentation: https://scikit-learn.org/stable/modules/tree.html

10. Decision Tree in Python: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

11. Naive Bayes Documentation: https://scikit-learn.org/stable/modules/naive_bayes.html

12. Bernoulli Naive Bayes in Python: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html

13. StratifiedKFold in Python: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

14. GridSearchCV https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

15. Heatmap https://seaborn.pydata.org/generated/seaborn.heatmap.html

16. SVM Implementation https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

17. KNN Implementation https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html