Rental Listing Enquiries: A Kaggle Challenge

Ameya Gamre, Jing Zhou,and Yili Zou

***Abstract*—Online rental systems are becoming increasingly popular over the last couple of years as it is getting really easy to find the apartments of our choice. This is based on the apartment listings on the websites and users expressing their interest in those listings. This interest can be based on a number of factors. Some of the important factors can be the price of that apartment, latitude and longitude of that place, or whether that apartment has certain amenities or not and many others. In this project, we try to predict the interest levels of apartments in New York city. The data we are using to base our predictions on are the rental listings from RentHop under a Two Sigma Competition.**

I. Introduction

Apartment rentals have been the norm of the hour and have become an important part of the real estate business. Online portals have gained a lot of attention over the years as it makes it easier for the users to go online and look for the apartments they need to rent based on the location, price, amenities and other criteria. RentHop is one such hosting where users can search for apartments in areas such as New York, Boston, Chicago and other metropolitan cities. We can use this data to calculate the correlation between different attributes so that customers can benefit from it by finding the apartment most suitable to their needs.

II. DataSet

The dataset has 49352 training examples and 74659 samples for testing. One Listing had the following features:

- bathrooms: number of bathrooms
- bedrooms: number of bathrooms
- building_id
- created
- description
- display_address
- features: a list of features about this apartment
- latitude
- listing_id
- longitude
- manager_id
- photos: a list of photo links. You are welcome to
- download the pictures yourselves from renthop's site, but they are the same as imgs.zip.
- price: in USD
- street_address
- interest_level: this is the target variable. It has 3 categories: 'high', 'medium', 'low'

III. Problem Definition and Algorithm

A. Task Definition

Prediction of rental listing interest levels on the apartment listings of New York City. Input is training and testing sets which are in JSON format. Output is the probability of each listing coming under low, medium and high.
The following is the plot of ListingID vs Interest Levels.

B. Algorithm Definition

We use three models for classification of each rental listing. These are Naive Bayes, Random Forest and XGBoost.

*1) **XGBoost**:* This is an introductory document of using the xgboost package in Python. xgboost is short for eXtreme Gradient Boosting package. It is an efficient and scalable implementation of gradient boosting framework. The package includes efficient linear model solver and tree learning algorithm. It supports various objective functions, including regression, classification and ranking. The package is made to be extendible, so that users are also allowed to define their own objectives easily. It has several features:

1) Speed:xgboost can automatically do parallel computation on Windows and Linux, with openmp. It is generally over 10 times faster than gbm.
2) Input Type:xgboost takes several types of input data:
   - Sparse Matrix
   - Dense Matrix
   - Data File:Local data files
   - xgb.DMatrix:xgboosts own class.Recommended.
3) Sparsity:xgboost accepts sparse input for both tree booster and linear booster, and is optimized for sparse input.
4) Customization:xgboost supports customized objective function and evaluation function
5) Performance:xgboost has better performance on several different datasets.

+++ Python Library Sklearn was used for model selection, preprocessing, calculating logloss and xgboost model was used. XGBoost: xgboost(trainX,trainY,testX,testY)

1) Define all parameters for the algorithm
2) Form matrices of training vectors and their classes and testing data and their classes
3) if testY is not Null: Create a WatchList of Training Set and Testing Set Train the model using parameters, training set and the watchlist.

4) else: train the model over parameters and training set only using the train method of xgboost
5) predict using testset using the predict method of xgboost
6) return prediction, model

*2) **Random Forest**:* Decision trees are a popular method for various machine learning tasks. because it is invariant under scaling and various other transformations of feature values, is robust to inclusion of irrelevant features. However, they are seldom accurate. In particular, trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e. have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

Tree bagging

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set X = x1, ..., xn with responses Y = y1, ..., yn, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For b = 1, ..., B:

1) Sample, with replacement, B training examples from X, Y; call these Xb, Yb.
2) Train a decision or regression tree fb on Xb, Yb.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x': or by taking the majority vote in the case of decision trees.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

From bagging to random forests The above procedure describes the original bagging algorithm for trees. Random forests differ in only one way from this general scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated. An analysis of how bagging and random subspace projection contribute to accuracy gains under different conditions. Typically, for a classification problem with p features, p (rounded down) features are used in each split.// Data: bathrooms, bedrooms, latitude, longitude, price, num_photos, num_features, num_description_words, created_year, created_month, created_day.

*3) **Naive Bayes**:* **Algorithm:**

This is an optimized version of Nave Bayes algorithm. Since in this dataset, some features like price, latitude, longitude are closed to continuous variable, each data points have distinct value, which makes it not suitable for Nave Bayes algorithm and takes too long to run.// **Simplified Psuedocode logic:**

Def feature_transformation_price()
Def feature_transformation_photos()
Def feature_transformation_latitude()
Def feature_transformation_longitude()
Def feature_transformation_feature()
Main()

1) Processing data from Json
2) Prepare data to hashtable, and extract necessary data features that we are going to use.
3) Use feature scaling and transformation technique to convert price, photos, latitude, longitude features to an alternative value. For example, use threshold to cut price value, use number of photos and features to replace its original value, use transformation function to convert latitude and longitude so that they only have 30ish distinct value across the entire dataset.
4) Calculate the prior probability for high, medium, low probability of interest level.
5) For each feature, calculate the conditional probability of each distinct value, respecting to high, medium, low condition.
6) Read another data set, extract features that we are going to use. And do the same feature transformation using the methods above.
7) Use log scale for priors and conditional probabilities, calculate the score for each listing, and convert the log scaled score back to probability as the probabilities that we need for output.
8) Generate prediction based on probability and compare to the correct label for accuracy. And outputting the probabilities to file.

IV. Experimental Evaluation

A. Methodology

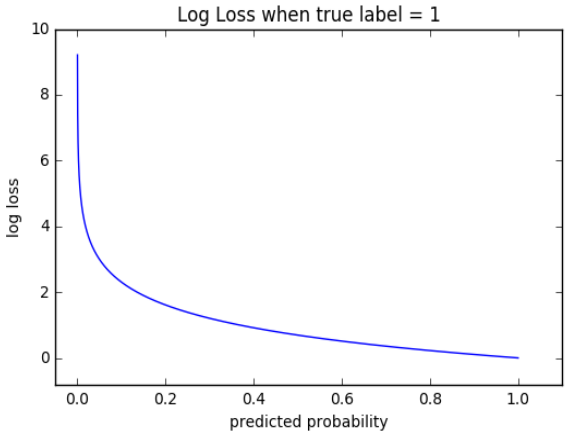
*1) **XGBoost**:* Some of the parameters used for Xgboost are:

- eta [default=0.3, alias: learning_rate] step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features. and eta actually shrinks the feature weights to make the boosting process more conservative. range: [0,1]
- max_depth [default=6] maximum depth of a tree, increase this value will make the model more complex / likely to be overfitting. range: [1,∞]

- objective [default=reg:linear] ”multi:softprob” –same as softmax, but output a vector of ndata * nclass, which can be further reshaped to ndata, nclass matrix. The result contains predicted probability of each data point belonging to each class.
- silent [default=0] 0 means printing running messages, 1 means silent mode.
- num_class(number of classes)
- eval_metric [default according to objective] evaluation metrics for validation data, a default metric will be assigned according to objective. User can add multiple evaluation metrics, for python user, remember to pass the metrics in as list of parameters pairs instead of map, so that latter ’eval_metric’ won’t override previous one. Eg. ”logloss”: negative log-likelihood, ”mlogloss”: Multiclass logloss.
- min_child_weight [default=1] minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. In linear regression mode, this simply corresponds to minimum number of instances needed to be in each node. The larger, the more conservative the algorithm will be. range: [0,∞]
- subsample [default=1] subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collected half of the data instances to grow trees and this will prevent overfitting. range: (0,1]
- colsample_bytree [default=1] subsample ratio of columns when constructing each tree. range: (0,1]
- seed [default=0]

Features used are:
bathrooms,bedrooms,latitude,longitude,price(numerical)
display_address,manager_id,building_id,street_address
(categorical)
Data was in JSON format and was read using pandas library. The numerical features were just manipulated using simple arithmetic. For the categorical features, we had to do preprocessing and encode those features including both train and test data sets. When label encoding we combine the train and test set before doing the encoding so that all like values have the same encoded values. For example, if the train set consisted of [a,c,d,e] and the test set has [a,b,c,d] you would have d encoded as 2 in the train set, while c is encoded as 2 in the test set. The model would then treat all the c’s in the test set as equal to d in the training set.
We have features column which is a list of string values. So we can first combine all the strings together to get a single string and then apply count vectorizer on top of it. Count Vectorizer converts a collection of text documents to a matrix of token counts.
We learn the vocabulary dictionary and return term-document matrix using transform. We then stack sparse matrices horizontally (column wise) . Now we stack both the dense and sparse features into a single dataset where dense features are the previous numerical and categorical features and the sparse features are the ones under the attribute features. We map the target attributes high, medium and low to 0,1 and 2 respectively. Now we split the training data into training and validation sets and do some cross validation. ’test-mlogloss’ will be used for early stopping. Now we build the final model and get the predictions on the test set. We have successfully trained the algorithm on the training set and have predictions on the test set according to low, medium, high with their probabilities.

Why LogLoss instead of Accuracy?
Accuracy is the count of predictions where your predicted value equals the actual value. Accuracy is not always a good indicator because of its yes or no nature. Log Loss takes into account the uncertainty of your prediction based on how much it varies from the actual label. This gives us a more nuanced view into the performance of our model. The graph below shows the range of possible log loss values given a true observation (isDog = 1).



As the predicted probability approaches 1, log loss slowly decreases. As the predicted probability decreases, however, the log loss increases rapidly. Log loss penalizes both types of errors, but especially those predications that are confident and wrong!

Logarithmic Loss, or simply Log Loss, is a classification loss function often used as an evaluation metric in kaggle competitions. Success in these competitions hinges on effectively minimising the Log Loss. Log Loss quantifies the accuracy of a classifier by penalising false classifications. Minimising the Log Loss is basically equivalent to maximising the accuracy of the classifier. In order to calculate Log Loss the classifier must assign a probability to each class rather than simply yielding the most likely class.

*2) **Random Forest**:* Here again, we used the logarithmic loss to evaluate result. Log loss, aka logistic loss or cross-entropy loss. In order to calculate Log Loss the classifier must assign a probability to each class rather than simply yielding the most likely class. Mathematically Log Loss is defined as It suits to multi-class tasks

$$l(y,p) = -ylog(p) - (1-y)log(1-p)$$

Average log loss for N instances is
$\frac{1}{N}\sum_{i=1}^{N} l(y_i, p_i)$ (1)

*3) **Naive Bayes**:* Since the output of result is probabilities regarding to 3 different labels, so the criteria we are evaluating is the confidence level of the probability, namely, how accurate would my prediction be if we use the result probabilities. Our independent variables will be the number of features we use for prediction, the threshold when we do the data transformation for price, latitude, and longitude.
Transformation function for price: price_value = int(price/threshold)
Transformation function for latitude: latitude_value = int((latitude -40)*10000/threshold)
Transformation function for longitude: longitude_value = int((longitude+74)*10000/threshold)
For photos and features, just convert them to the number of photos and number of features.
Change different values for these threshold, and use more or less features, see how accuracy change would respond to it. My hypothesis is more features used, more accurate is the model, and the accuracy will change when use different threshold, and there is an optimal range of threshold.

B. Results

*1) **Naive Bayes**:* When price threshold is 400, latitude threshold is 300, longitude threshold is 300. Try different number of features used.

Feature used	Bathrooms, Bedrooms, price	Bathrooms, Bedrooms, Price, photos	Bathrooms, Bedrooms, Price,photos, latitude longitude	Bathrooms, Bedrooms, Price, photos, latitude longitude, features
Prediction Accuracy	0.543	0.558	0.665	0.667

When using data features bathrooms, bedrooms, price, photos, latitude, longitude, and features. Let price threshold be 400, try different threshold for latitude and longtitude

Threshold of latitude and longitude	100	300	500	700	900
Prediction Accuracy	0.64	0.669	0.658	0.647	0.635

Let latitude and longitude threshold be 400, try different threshold for price. Our hypothesis is supported. The more features we use, the more accurate is the model. The accuracy indeed changes when we use different thresholds, however the difference is not significant. The number of features impact most in this model. I noticed the running speed changes a lot for different threshold. This is because when threshold is too small, there will be too many distinct value for those features,

Threshold of price	100	300	500	700	900
Prediction Accuracy	0.664	0.665	0.664	0.662	0.655

therefore it takes longer to execute. I would sacrifice a little accuracy for the sake of running speed, and try to use more features possible.

2) Random Forest and XGBoost: The logloss values for Random forest and XGboost are in the table below:
For XGBoost, we used kFold CrossValidation and divided the dataset into 5 folds and we observed the logloss values of the intermediate training steps. The model will train until the validation score stops improving. Validation error needs to decrease at least every early_stopping_rounds to continue training.

```
Will train until test error hasn't decreased in 20 rounds.
[0] train-mlogloss:1.039784 test-mlogloss:1.041510
[1] train-mlogloss:0.991683 test-mlogloss:0.994820
[2] train-mlogloss:0.950237 test-mlogloss:0.954704
[3] train-mlogloss:0.910744 test-mlogloss:0.916360
[4] train-mlogloss:0.877119 test-mlogloss:0.884209
[5] train-mlogloss:0.846378 test-mlogloss:0.854590
[6] train-mlogloss:0.821941 test-mlogloss:0.831212
[7] train-mlogloss:0.799380 test-mlogloss:0.809554
[8] train-mlogloss:0.778433 test-mlogloss:0.789901
[9] train-mlogloss:0.758998 test-mlogloss:0.771475
[10]    train-mlogloss:0.743267 test-mlogloss:0.756815
[11]    train-mlogloss:0.728783 test-mlogloss:0.743287
[12]    train-mlogloss:0.716622 test-mlogloss:0.732233
[13]    train-mlogloss:0.705494 test-mlogloss:0.721952
[14]    train-mlogloss:0.694856 test-mlogloss:0.712177
[15]    train-mlogloss:0.685141 test-mlogloss:0.703540
[16]    train-mlogloss:0.675977 test-mlogloss:0.695158
[17]    train-mlogloss:0.666923 test-mlogloss:0.686980
[18]    train-mlogloss:0.659421 test-mlogloss:0.680168
[19]    train-mlogloss:0.651546 test-mlogloss:0.673299
[20]    train-mlogloss:0.646086 test-mlogloss:0.668534
[21]    train-mlogloss:0.640589 test-mlogloss:0.663941
[22]    train-mlogloss:0.634956 test-mlogloss:0.659270
[23]    train-mlogloss:0.629039 test-mlogloss:0.654011
[24]    train-mlogloss:0.623982 test-mlogloss:0.649389
[25]    train-mlogloss:0.619367 test-mlogloss:0.645655
[26]    train-mlogloss:0.614638 test-mlogloss:0.641577
[27]    train-mlogloss:0.611298 test-mlogloss:0.638823
```

```
[312]   train-mlogloss:0.370701 test-mlogloss:0.553488
[313]   train-mlogloss:0.370311 test-mlogloss:0.553403
[314]   train-mlogloss:0.369868 test-mlogloss:0.553470
[315]   train-mlogloss:0.369438 test-mlogloss:0.553489
[316]   train-mlogloss:0.369118 test-mlogloss:0.553584
[317]   train-mlogloss:0.368561 test-mlogloss:0.553685
[318]   train-mlogloss:0.368209 test-mlogloss:0.553779
[319]   train-mlogloss:0.367520 test-mlogloss:0.553576
[320]   train-mlogloss:0.367142 test-mlogloss:0.553599
[321]   train-mlogloss:0.366634 test-mlogloss:0.553509
[322]   train-mlogloss:0.366197 test-mlogloss:0.553526
[323]   train-mlogloss:0.365750 test-mlogloss:0.553582
[324]   train-mlogloss:0.365164 test-mlogloss:0.553503
[325]   train-mlogloss:0.364784 test-mlogloss:0.553620
[326]   train-mlogloss:0.364518 test-mlogloss:0.553748
[327]   train-mlogloss:0.364065 test-mlogloss:0.553760
[328]   train-mlogloss:0.363657 test-mlogloss:0.553701
[329]   train-mlogloss:0.363257 test-mlogloss:0.553767
[330]   train-mlogloss:0.362943 test-mlogloss:0.553800
[331]   train-mlogloss:0.362571 test-mlogloss:0.553766
[332]   train-mlogloss:0.362152 test-mlogloss:0.553720
[333]   train-mlogloss:0.361849 test-mlogloss:0.553798
Stopping. Best iteration:
[313]   train-mlogloss:0.370311 test-mlogloss:0.553403

[0.55379868325891934]
```

	LogLoss
Random Forest	0.632
XGBoost	0.553

V. Related Work

As mentioned previously, this data has been gathered from public rental listings on renthop.com. The data obtained thus

is of a heterogeneous nature, containing numerical as well as non-numerical fields. A similar problem was solved, the work of which is published in the paper.[1] It is a classic example of a multiclass classification problem. Binary classifiers such as Neural Networks, Decision Trees, SVM, etc can be extended to address these kind of problems. Our task in essense is fairly similary to rating prediction tasks,thus,inspiration can be taken from work done previously on such problems. Work has been done on Text Based Rating Predictions from Beer and Wine Reviews, which uses review text data majorly for its prediction and uses Naive Bayes while predicting rating as categorical vari- able instead of numerical value similar to our task. An other example of predicting movie user rating with Imdb Attributes, which uses linear classification and neural networks to solve the problem. A study for prediction movie ratings, compares results of Random Forests and Support Vector Machines for the prediction task. Gradient Boosting has been fairly successful at solving prediction tasks. One author, in his thesis predicts rating on Netflix Prize Dataset using gradient boosting machines. In a Kaggle article, author explains Gradient Boosting which has been successful in a variety of Kaggle tasks and can be generalized to many problems.

VI. Future Work

We can aim to improve the accuracy of the models by adding and processing more features and use more sophisticated transformation methods for features such as photos, description and features,etc. Image processing and feature extraction was not done in this project but it can be done in the future using techniques such as Convolutional Neural Networks.

VII. Conclusion

Random Forest and XGBoost have better accuracy compared to Naive Bayes. However, Naive Bayes runs faster and the accuracy can be improved by adding more features in the future.

References

[1] A.Patankar, K. Uppal, N. Kulkarni, K.Khalid *Two Sigma Connect: Predicting Interest Level of Rental Listings*, 2017.