Rental Listing Inquiries

Yang Li, Lupeng Xing, Xiaofan Jin

Data Mining - Final Project

Northeastern University

April 28, 2017

Overview

Most people had rental experiences. In order to live into an ideal apartment, we often draw support from housing rental websites. Some agents used these websites to post advertisements. However, different advertisements can bring different numbers of click. We are all interested by this phenomenon, so we decided to find out the reason of it.

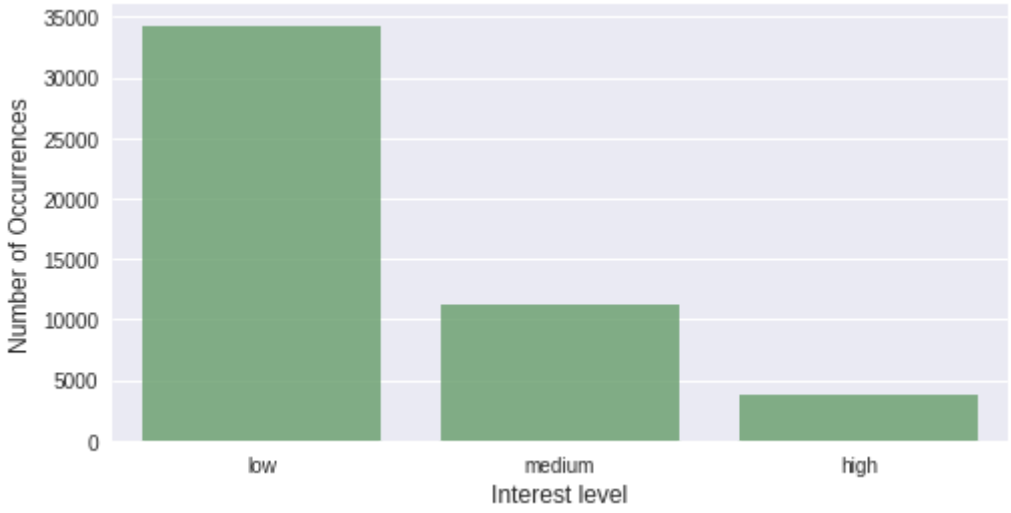
Two Sigma has launched a competition on Kaggle to predict rental listing popularity[1] .It provides us a very accuracy dataset, which contains about 40,000 instances. Data comes from an apartment listing website, called renthop.com. These apartments are located in New York City.

Doing so will help agents predict the popularity of a rental listing based on various features, at the same time, it will allow agents better handle fraud control, identify potential listing quality issues, and understand renters' needs and preferences.

Original Data Analysis

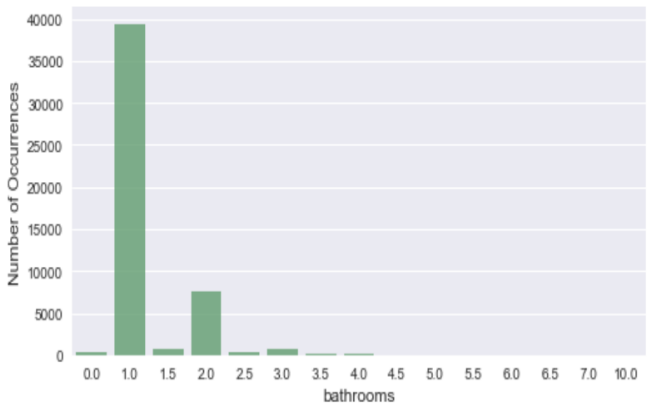
Each instance has 13 features, bathrooms, bedrooms, latitude, longitude, price, photos, description, created time, listing id, manager id, building id, address, and interest level.

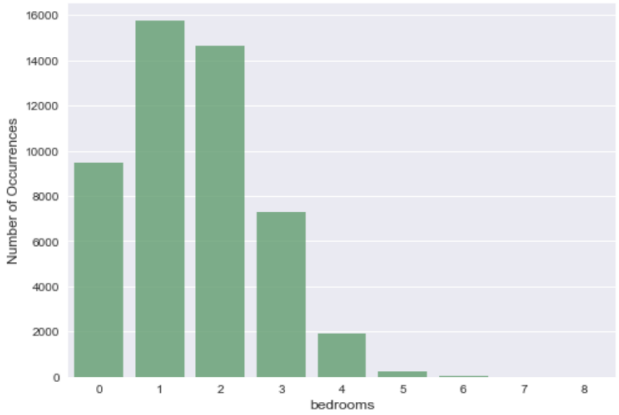
Interest level is the target variable Y. It means the numbers of click, and it has three classes, low, medium, and high, which is considered to have a positive relation with user's interest level.



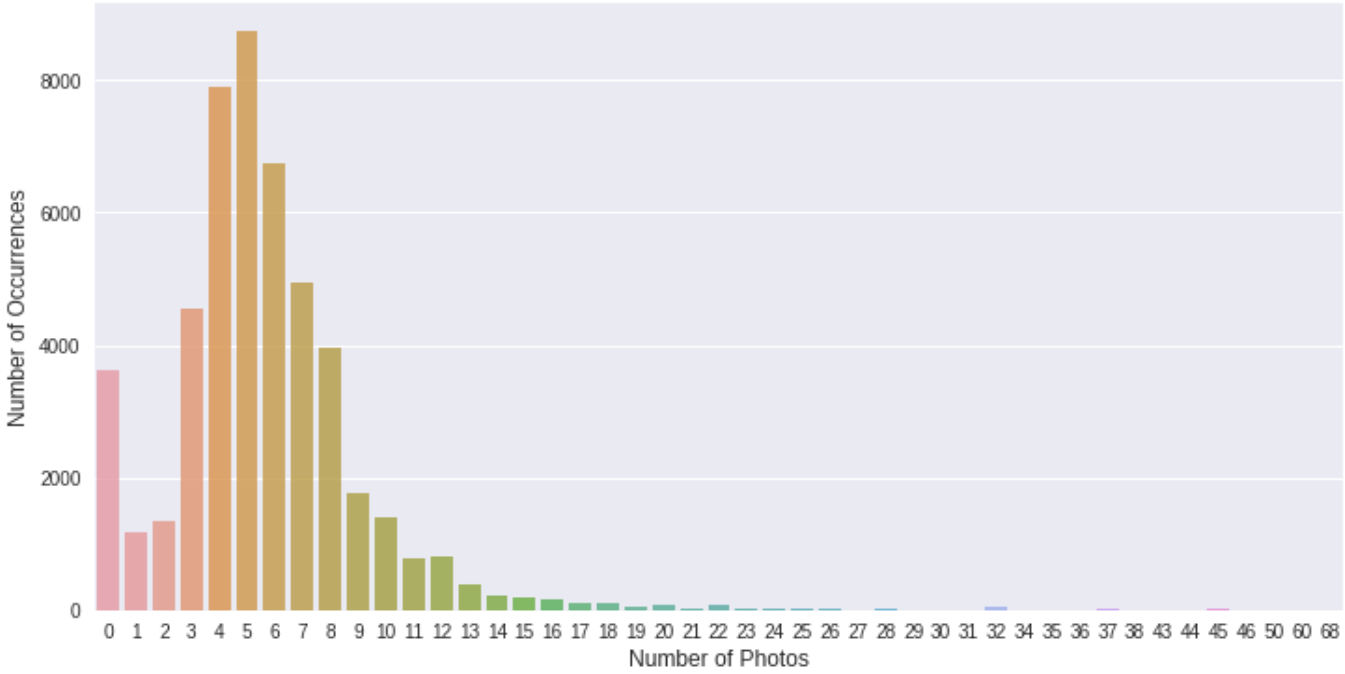
As we can see, the dataset is unbalanced, there are almost 35,000 instances with low interest level, but only lower than 5000 instances with high interest level.

Quantity distribution histograms of bathroom and bedrooms:





For photos feature, in our project, we estimate the amount of photos. It is consistent with common sense that the more photos the advertisement provides, the more likely for user to click the link.



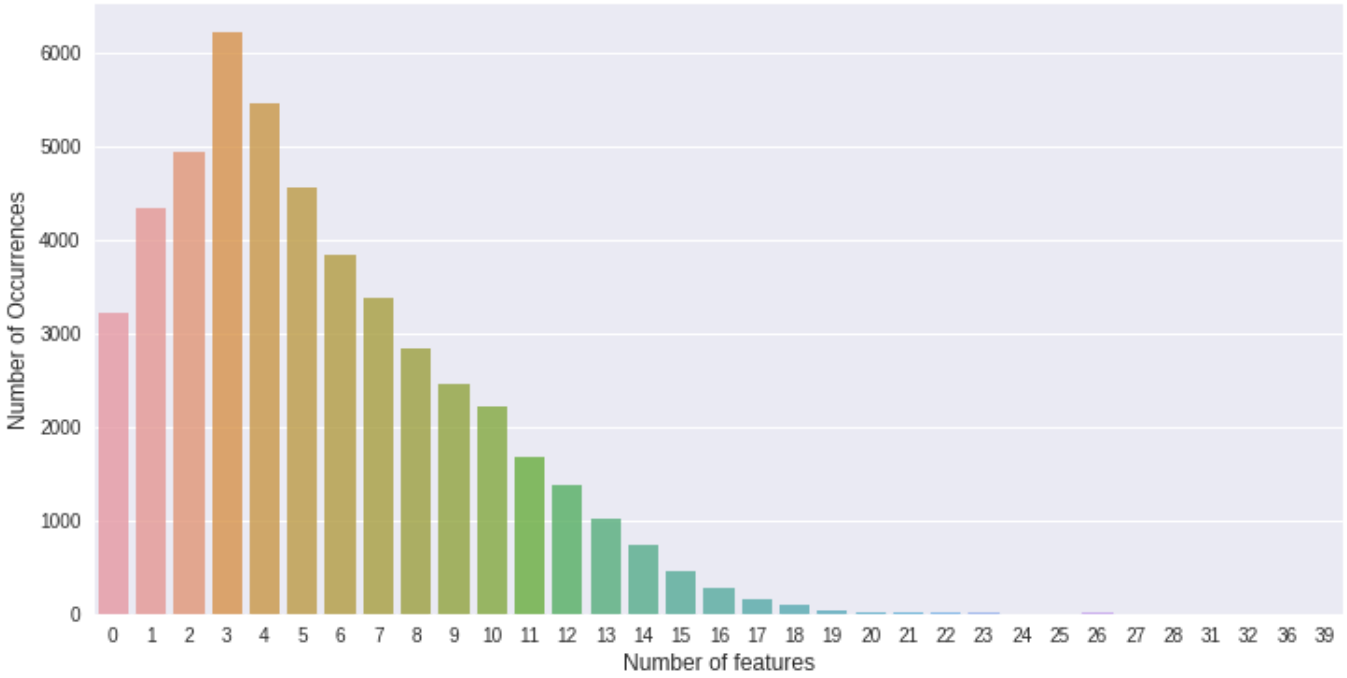
It is interesting to find out there are about 3,800 advertisements, which do not have any photos posted.

There is a feature in our data called “feature”, it is a set of key words which describes the apartment. For a json format example,

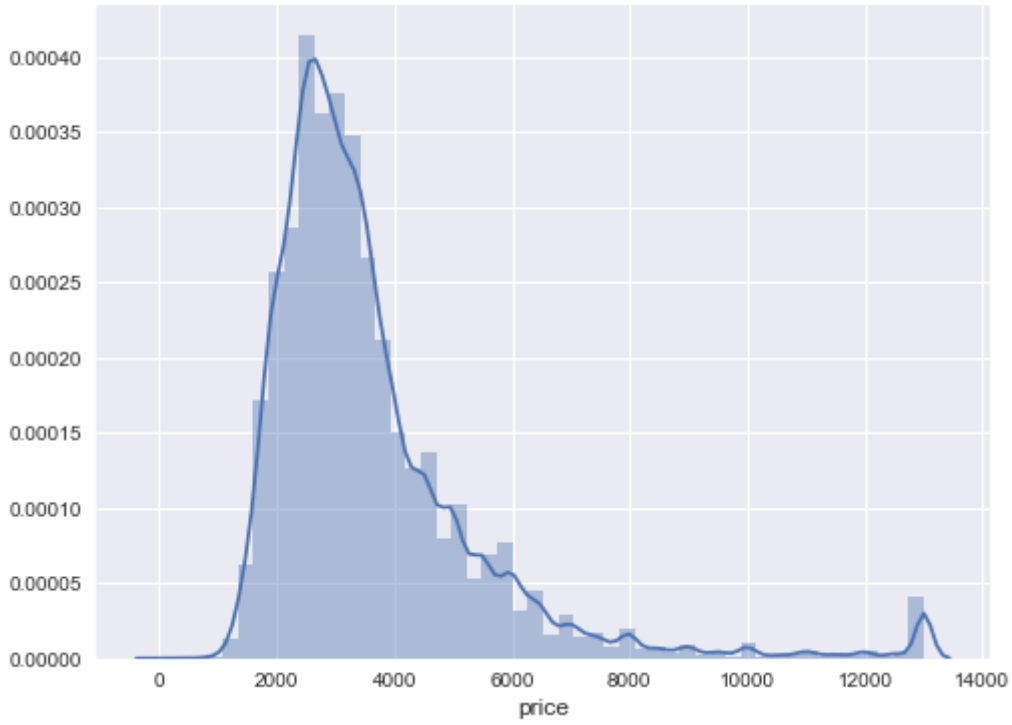
```
{"feature": "Doorman, Elevator, Fitness Center, Cats Allow"}
```

it shows the apartment has doorman, elevator, fitness center in the building, and cats are allowed.

Although, there would be different key words of description, the degree of understanding of the house is likely to be determined by the features’ amount.



Price, which is considered to be the most important reason for users to decide weather to click the link or not, needs a graph that shows the distribution of the price.



As the graph shown, most of the apartments have the price from $2,000 to $4,000.

Feature Engineering

After careful analysis of original data, we had a basic understanding of significance of different attributes, but we still need an experiment to justify our assumption. Before our experiment, we need feature engineering to convert our attributes to proper features for training process.

For numeriacal attributes, including number of bathromms and bedrooms, location and price, we don't need pre process and would just keep these as features.

For photos of each house, we currently do not deal with the detail of them since that would be a time-comsuming training process and we would just use the number of photos for each house as a feature, since ususlly it's assumed that plenty photos would provide enough information about the house and it's likely to be interested by users. So number of photos would be a positive factor.

For descriptive things , including features and description words, we also decide to just use the number of these attributes as our features, rather than complex natural language analysis, due to the limited training time. They're also assumed to have positive effect.

For information created date, we assume there would be some seasonal and historical effect on users' interest level, so we decide to extract different parts, year, month, day, hour from the original date attribute and treat these features individually.

For categorical attributes, it's hard to use them without label encoding. So for these four attributes, including display_address, manager_id, building_id, listing_id, we transform these into discrete numeric values.

```
categorical = ["display_address", "manager_id", "building_id", "street_address"]
for f in categorical:
        if train_df[f].dtype=='object':
            lbl = preprocessing.LabelEncoder()
            lbl.fit(list(train_df[f].values) + list(test_df[f].values))
            train_df[f] = lbl.transform(list(train_df[f].values))
            test_df[f] = lbl.transform(list(test_df[f].values))
            features_to_use.append(f)
```

For "features" attribute, which is a list of string values that act as the keywords of the house, we first combine all the strings together, and then apply count vectorizer on top of it.

```
train_df['features'] = train_df["features"].apply(lambda x: "
".join(["_".join(i.split(" ")) for i in x]))
test_df['features'] = test_df["features"].apply(lambda x: "
".join(["_".join(i.split(" ")) for i in x]))
tfidf = CountVectorizer(stop_words='english', max_features=200)
```

Model Selection

From the data analysis, we know it's a typical multivariable classification problem, and it's a classical scenario for decision tree models. But naive decision tree algorithm won't exceed expected performance and accuracy, so it's a nature idea to combine boosting method to improve the training accuracy. Therefore, we decide to try three different algorithms, including AdaBoost, Random Forests, and Xgboost.

AdaBoost[2], short for "Adaptive Boosting", is a machine learning meta-algorithm and can be used in conjunction with many other types of learning algorithms to improve their performance. AdaBoost with decision trees as the weak classifier is quite commonly used and is sensitive to noisy data and outliners.

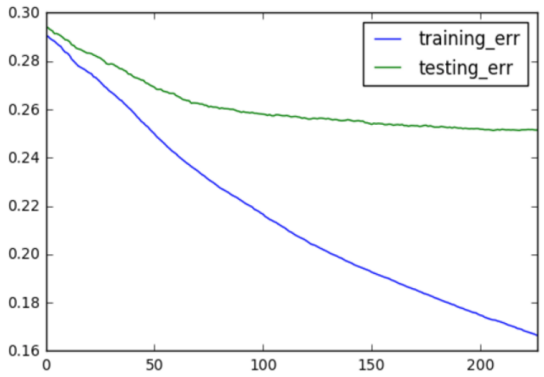
Random Forests[3] are an ensemble learning method for classification operated by constructing a multitude of decision trees at training time and outputting the class that is the mean prediction of the individual trees.

We use Xgboost[4] as the main framework to train the classifier. Xgboost, which is widely used in Kaggle competition, is an optimized distributed gradient boosting labrary. It's developed by Tianqi Chen mainly and supports different programming languages including Python. The main advantages include highly efficiency and flexiblity. It implements machine learning algorithms under the Gradient Boosting framework. Xgboost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. Therefore, the main advantages of Xgboost are higher efficiency and accuracy.

Training

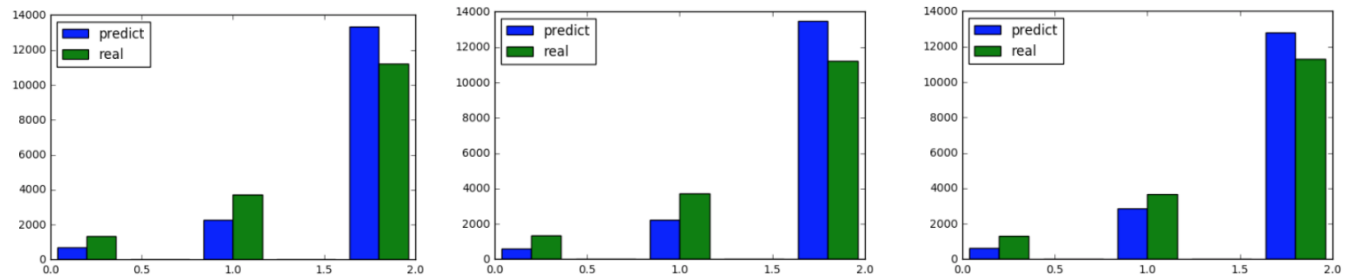
We split our data into two parts, 67% of data will be used for training the model and 33% will be used for testing. We trained Random Forest classifier, AdaBoost classifier, and XGBoost classifier respectively and then fit the testing data in each one to get prediction.

Take XGBoost for example, we need to figure out the best boosting rounds to avoid both underfitting and overfitting. In order to do that, we use cross validation method. We first split the dataset into 5 folds. In each round of boosting, we iteratively use 4 to train and use 1 to test, and get both the testing error rate and training error rate for each boosting round. As you can see from below graph, both testing error and training error are decreasing with the boosting rounds. The testing error stops improving at about 200 rounds. So we think 200 rounds should be a proper amount.



Testing

Below graphs are the predictions from each classifier, including AdaBoost, Random Forests and Xgboost.



From the above graph(high = 0, medium = 1, low = 2), it's observed more instances are predicted to be "low" compared with actual instances.

Evaluation

To evaluate the performance of our models, we will use confusion matrix, Precision and Recall Curve and ROC.

Actual\Predicted	high	medium	low
high	347	487	492
medium	228	1084	2420
low	102	718	10409

AdaBoost

Actual\Predicted	high	medium	low
high	319	479	528
medium	189	1099	2444
low	73	661	10495

Random Forests

Actual\Predicted	high	medium	low
high	355	649	294
medium	196	1429	2073
low	58	808	10425

XGBoost

	precision	recall	f1-score	support
high	0.26	0.51	0.35	677
medium	0.29	0.47	0.36	2289
low	0.93	0.78	0.85	13321
Avg/total	0.81	0.73	0.76	16287

AdaBoost

	precision	recall	f1-score	support
high	0.24	0.55	0.33	581
medium	0.29	0.49	0.37	2239
low	0.93	0.78	0.85	13467
Avg/total	0.82	0.73	0.77	16287

Random Forests

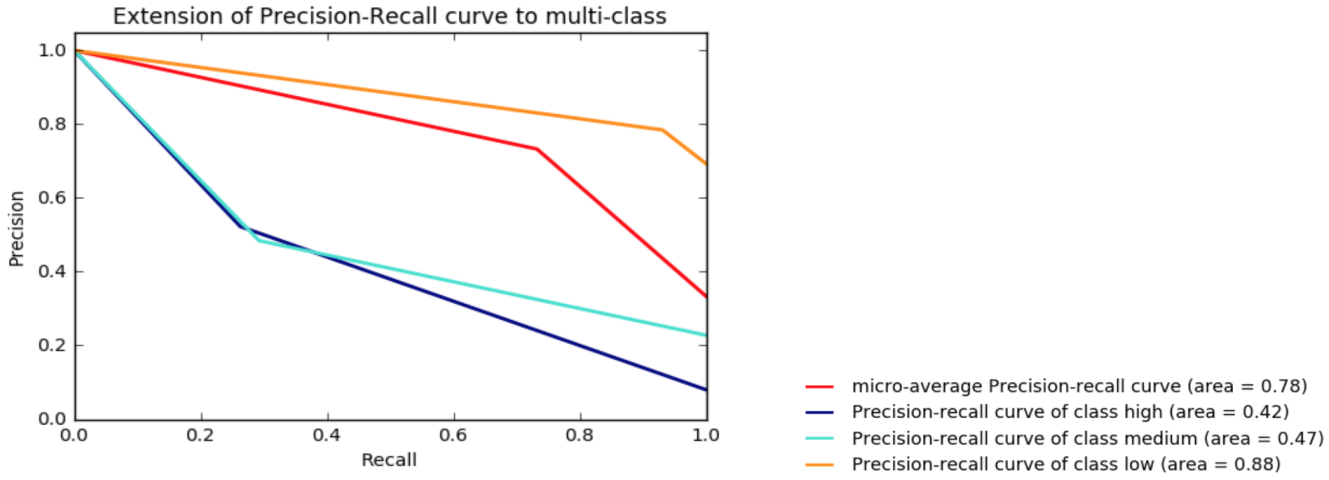
	precision	recall	f1-score	support
high	0.27	0.58	0.37	609
medium	0.39	0.50	0.43	2886
low	0.92	0.81	0.87	12792
Avg/total	0.80	0.75	0.77	16287

XGBoost

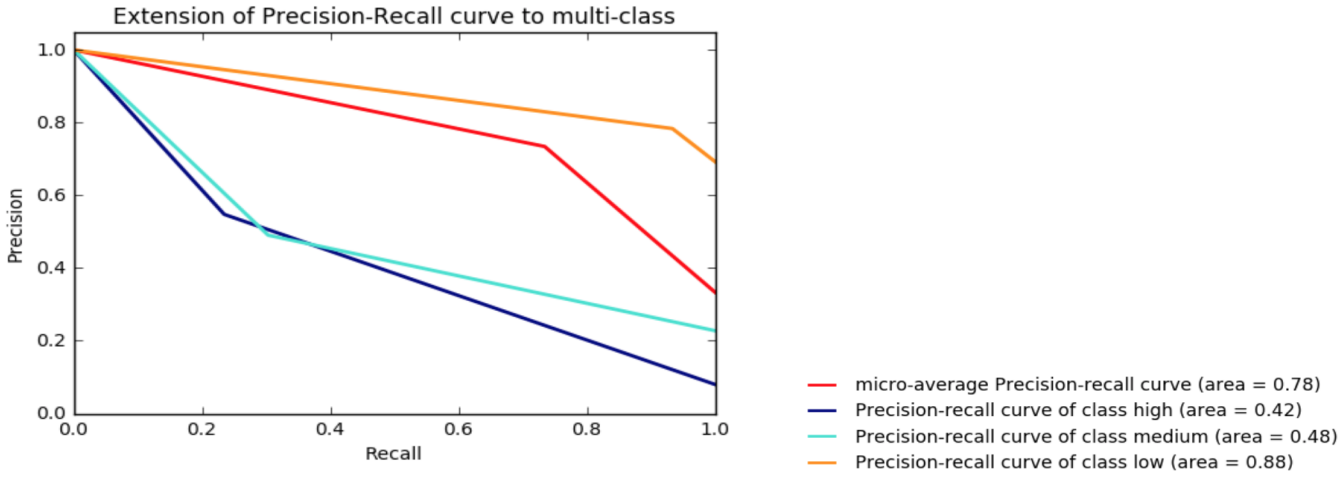
We get the final evaluation score by comparing the prediction with actual result of testing data.

We can see from the Confusion Matrix that all 3 classifiers have difficulty in predicting the high and medium classes. The reason is the imbalanced data set. Classifiers get trained pretty well with low class because there are more instances belonging to the low class.

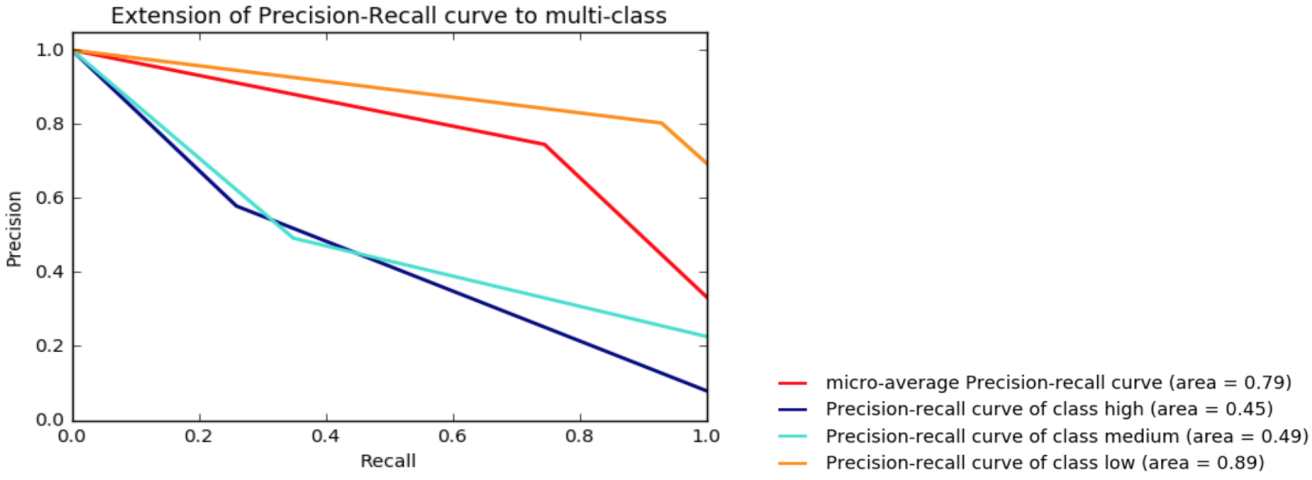
We should try to find solution to overcome the imbalanced data. SMOTE is a good technique for this scenario.



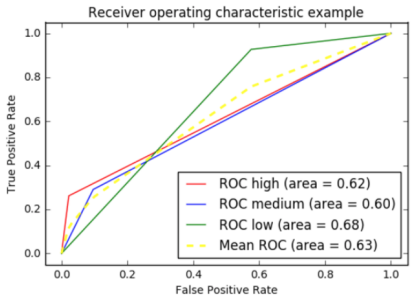
AdaBoost



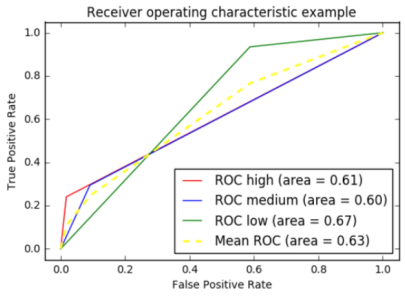
Random Forest



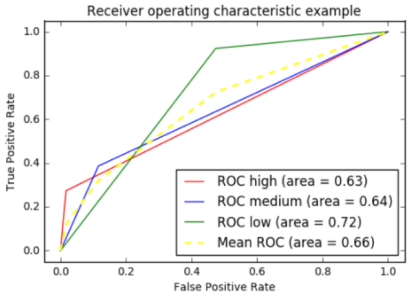
XGBoost



AdaBoost



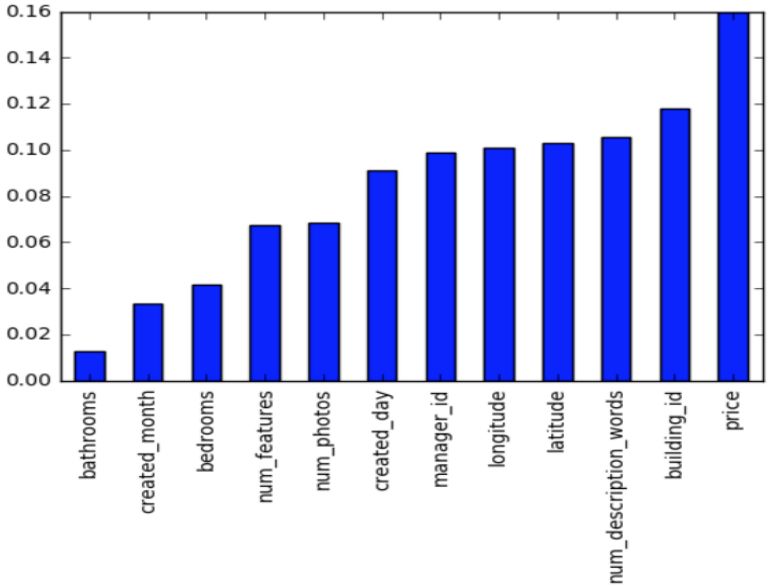
Random Forests



XGBoost

From the Precision-Recall curve and ROC curve we can conclude that the average predict precision of all 3 classifies is fine. They all perform well when predicting an instance that belongs to low class. They have some problems predicting high and medium instances. The XGBoost classifier has a higher precision than the other two.

Summary



We get an internal view of the importance of each feature from Random Forest model. Price, building id, number of description words, and location are the more important than other features. The useful information is in order to make a rental housing popular, the agent manager should probably consider a reasonable price and describing more.

Potential Improvement

As we can see in above analysis, our model doesn't perform well for instances that belong to high and medium classes. The imbalance dataset prevents our model from predicting precisely. We have two solutions for this problem. First, remove some instances that belong to low class. Second, create more instances that belong to high and medium classes. SMOTE is a proper technique here.

Besides, we can take advantage of image features to train our models. It is obvious that a clean and well decorated house will be more popular than an under decorated one, given other conditions the same. In the future, we should consider including some image processing in our model.

Reference

[1]https://www.kaggle.com/c/two-sigma-connect-rental-listing-inquiries/data

[2]https://en.wikipedia.org/wiki/AdaBoost

[3]https://en.wikipedia.org/wiki/Random_forest

[4]https://en.wikipedia.org/wiki/Xgboost