Machine Learning Engineer Nanodegree

Capstone Report

Swetha Ganapathi Raman

Mar 8th 2017

I. DEFINITION

Project overview

Real state companies like Redfin have become very popular in the recent years, mainly due to using data science/Machine learning/predictive analytics techniques to accurately estimate the price of a home. It would be very interesting to use data science/ ML techniques for the rental market as well.

Renthop is a rental listing company that uses machine learning to make apartment search smarter by ranking rental listings by quality. This makes it easier for people looking to rent homes to find a high quality rental listing quickly instead of having to sift through all the listings, especially in hot rental markets like San francisco bay area, NYC etc.

The capstone project I am choosing for MLND is a current challenge in Kaggle posted by Renthop - Two Sigma Connect : Rental Listing Inquiries
https://www.kaggle.com/c/two-sigma-connect-rental-listing-inquiries

The objective of this challenge is to predict the number of inquiries a rental listing will receive based on listing's creation date and other features.

Data source for the problem is provided by Renthop.
https://www.kaggle.com/c/two-sigma-connect-rental-listing-inquiries/data

The dataset has the following files:
Train .json - the training set
Test.json - the test set
Images_sample.zip - listing images organized by listing_id (a sample of 100 listings)

Problem statement

This problem features rental listing data of apartments in New York city from Renthop. Given the data from the rental listing like creation date, text description, number of bedrooms, bathrooms, price and other features, the objective is to predict the popularity of a new listing. The popularity of a rental listing is classified as 'high', 'medium' or 'low'.

In machine learning, this is a classification problem and the objective here is to predict the probability of each class (high interest, medium interest, low interest) for a rental listing. The

target variable ‘interest_level’ is defined by the number of inquiries a listing has in the duration that the listing was live on the site.

As stated by Renthop, “This prediction will help Renthop handle fraud control, identify potential listing quality issues and allow owners and agents to better understand renter’s needs and preferences.”

For this problem, I intend to feed the input features from the training set to a couple of Supervised classifier algorithms to see which produces the most accurate output class (High, medium, low), measured by multi class log loss. The model which has the lowest multiclass log loss would be a good solution.

Evaluation metrics

The model will be evaluated using multi class logarithmic loss.

$$Log\ loss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} Y ij\ log(P ij)$$

Where N = number of listings in the test set

M = number of class labels (3 classes)

Log is the natural logarithm

Yij =1 if observation i belongs to class j and 0 otherwise

Pij = Predicted probability that observation i belongs to class j

Multi class log loss defined by Kaggle
https://www.kaggle.com/wiki/MultiClassLogLoss

The best model will be the one that has the lowest log loss.

I chose Log loss as the metric in this problem because, rather than simply using the accuracy of the model as a metric, I would like to make sure that the model is also penalised for making incorrect classifications. For example, if the model predicts a rental listing as ‘Low’ interest when it is actually a ‘High’ interest listing or vice versa, the model needs to be penalised. Since Renthop is interested in accurately predicting if a rental listing will receive a high interest, misclassified predictions will be detrimental to Renthop. A metric that captures this accurately is log loss.

Multi class log loss punishes the classifiers which are confident about an incorrect prediction. In the above equation, if the class label is ‘high’ (i.e the instance is from that class), and the predicted probability is near to 1 (classifier predictions are correct), then the loss is really low as log(x) --> 0 if x -->1. So, this instance contributes to a small amount of loss to the total loss. If this occurs for every single instance, i.e. the classifications are accurate, then the total loss will also approach 0.

On the other hand, if the class label is ‘high’ (the instance is from that class) and the predicted probability is close to 0 (the classifier is confident in its mistake), the loss can approach to infinity theoretically, as log (0) is undefined.

II. ANALYSIS

Data Exploration

The rental listing dataset provided by Renthop has train and test data.

```
Train rows: 49352
Train columns: 15
Test rows: 74659
Test columns: 14
```

The training data has 49352 entries with 15 features. The test data has 74659 entries with 14 features.

Here is a sample data from the training set

```
       bathrooms      bedrooms       building_id                         created
10     1.5            3              53a5b119ba8f7b61d4e010512e0dfc85    2016-06-24 07:54:24
description                                        display_address      features interest_level
A Brand New 3 Bedroom 1.5 bath ApartmentEnjoy ...  Metropolitan Avenue   []      medium
latitude       listing_id     longitude      manager_id
40.7145        7211212        -73.9425       5ba989232d0489da1b5f2c45f6688adc
photos                                             price   street_address
[https://photos.renthop.com/2/7211212_1ed4542e...  3000    792 Metropolitan Avenue
```

The different features in the dataset are as below. The feature names are self explanatory.We notice that the features are of multiple data types like numeric, text, categorical, image etc. The target feature is ‘interest_level’.

```
bathrooms          float64    - Number of bathrooms
bedrooms             int64    - Number of bedrooms
building_id         object    - Building ID
created             object    - Date and time of creation
description         object    - Text description of apartment listing
display_address     object    - Short address of apartment listing
features            object    - list of features the apartment has
interest_level      object    - Target feature with 3 classes: low,medium,high
latitude           float64    - latitude of apartment location
listing_id           int64    - Renthop listing ID
longitude          float64    - longitude of apartment location
manager_id          object    - Manager ID
photos              object    - list of photo links
```

```
price                int64    - rent price in USD
street_address      object    - Street address of apartment
dtype: object
```

Target feature: interest_level
Numeric features: bathrooms, bedrooms, latitude, listing_id, longitude, price
Categorical features: building_id, description, display_address, features, manager_id

Here is the preliminary statistics on the numeric features.

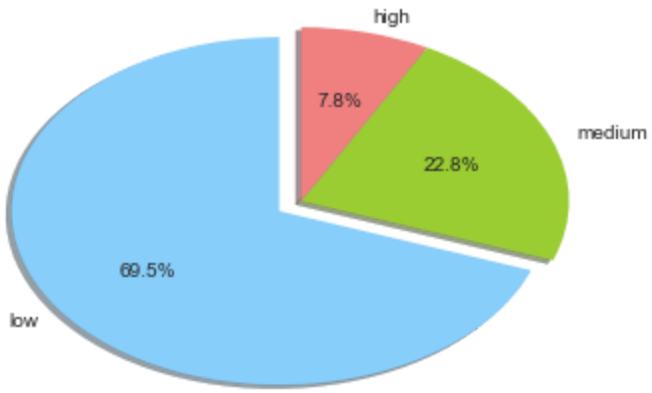
```
train.describe()
```

	bathrooms	bedrooms	latitude	listing_id	longitude	price
count	49352.00000	49352.000000	49352.000000	4.935200e+04	49352.000000	4.935200e+04
mean	1.21218	1.541640	40.741545	7.024055e+06	-73.955716	3.830174e+03
std	0.50142	1.115018	0.638535	1.262746e+05	1.177912	2.206687e+04
min	0.00000	0.000000	0.000000	6.811957e+06	-118.271000	4.300000e+01
25%	1.00000	1.000000	40.728300	6.915888e+06	-73.991700	2.500000e+03
50%	1.00000	1.000000	40.751800	7.021070e+06	-73.977900	3.150000e+03
75%	1.00000	2.000000	40.774300	7.128733e+06	-73.954800	4.100000e+03
max	10.00000	8.000000	44.883500	7.753784e+06	0.000000	4.490000e+06

We see that there are no missing values in the numeric features. We can further explore the data by plotting some charts and distributions.

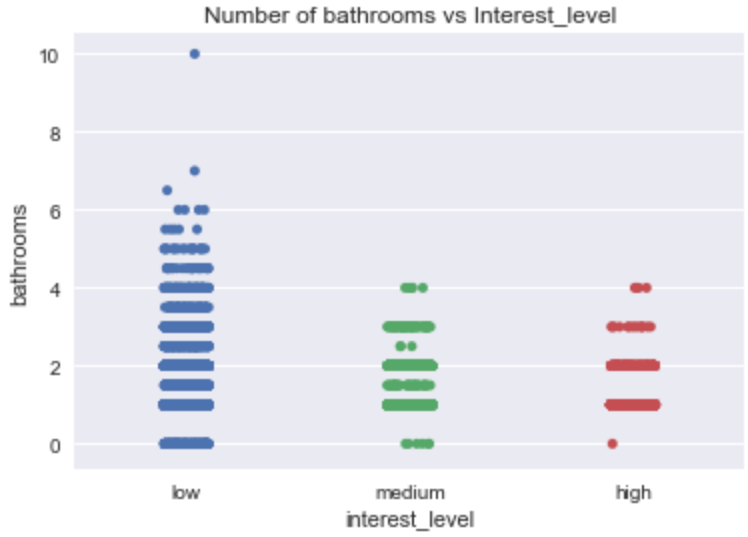
Exploratory Visualization
Feature: Interest level

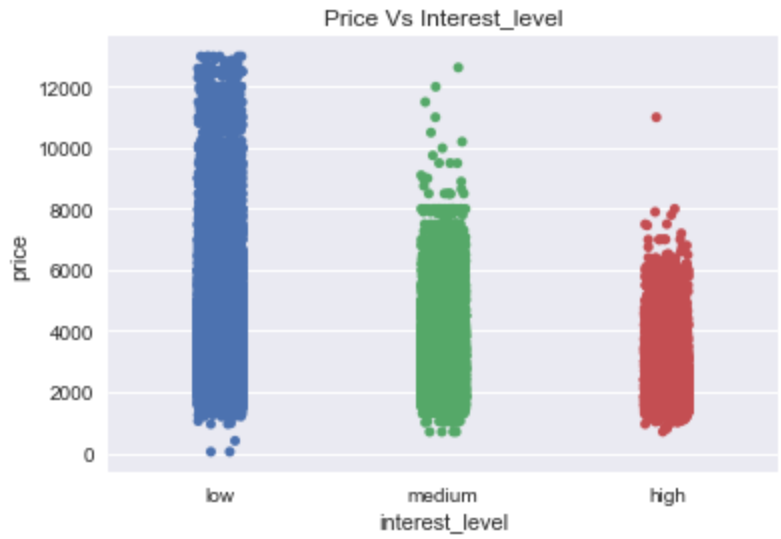
```
plt.pie(interest, explode=[0.1,0,0],
        labels=['low', 'medium', 'high'],
        colors= ['lightskyblue', 'yellowgreen', 'lightcoral'],
        autopct='%1.1f%%', shadow=True, startangle=90)
plt.show()
```



A majority (69.5%) of the rental listings have 'low' interest level, followed by 22.8% 'medium' interest level and only 7.8%'high' interest level.

Visualization of Bedrooms, bathrooms and price vs Interest_level





Some observations from visualization and data exploration

- **`Interest_level:`** There is a reasonable imbalance in classes with 69.5% having low interest and 7.8% high interest.
- **`Bedrooms :`** The distribution of number of bedrooms across different interest levels looks similar. Most of the medium and high interest listings have 0-4 bedrooms.
- **`Bathrooms:`** Most of the medium and high interest listings have 1-3 bathrooms.
- **`Price:`** Most of the medium and high interest listings have prices in the range $1000− $6000. The distribution of Price is right skewed. Hence a log transform is done and log_price is used as input feature
- **`created:`** The training and test dataset has rental listing data from April - June 2016

Benchmark model

For the benchmark model, I plan to use a model that always predicted that the listing has a 'low' interest, i.e. the output class is 'low'. I chose this because, "Low" is the most frequent output class in the training data.

Renthop has provided a sample_submission.csv output file along with the test and train dataset. This file has the output for the test data with the output class "Low" with the highest probability for all listings.

This is a good benchmark data to use. I would like my model to predict better than this benchmark rather than just predicting all listings to have "low" interest level and same probabilities.

```
y_benchmark = pd.read_csv("~/Documents/Kaggle/Twosigma/sample_submission.csv")
Y_benchmark.shape
(74659, 4)
```

```
y_benchmark.head(5)
```

	high	medium	low
0	0.077788	0.227529	0.694683
1	0.077788	0.227529	0.694683
2	0.077788	0.227529	0.694683
3	0.077788	0.227529	0.694683
4	0.077788	0.227529	0.694683

Benchmark log loss
Log loss for the benchmark model = 0.7905

This score has been obtained by submitting the sample_submission file in Kaggle. My objective is to come up with a model that has a log loss lesser than the benchmark.

Algorithms and Techniques
I plan to use two supervised classifier models for comparison.

Supervised classifiers
The two supervised classifiers I chose are

1. Logistic regression
2. Random Forest

Logistic Regression
I chose Logistic regression as it is my first go to algorithm of choice for classification problems. It is a well behaved simple classification algorithm and very robust to noise. It also gives probabilities associated with the classes. I would like to start with this simple algorithm and then go for more complex models.

Random Forest
The second algorithm I chose is Random forest. It is a popular algorithm for classification problems and has high accuracy. I chose this algorithm because it is known to work well with large data as well as inputs with multiple data types, as in our case. Also, it does not suffer from overfitting.

III. METHODOLOGY

Data Preprocessing
I did the following data preprocessing steps

Missing data

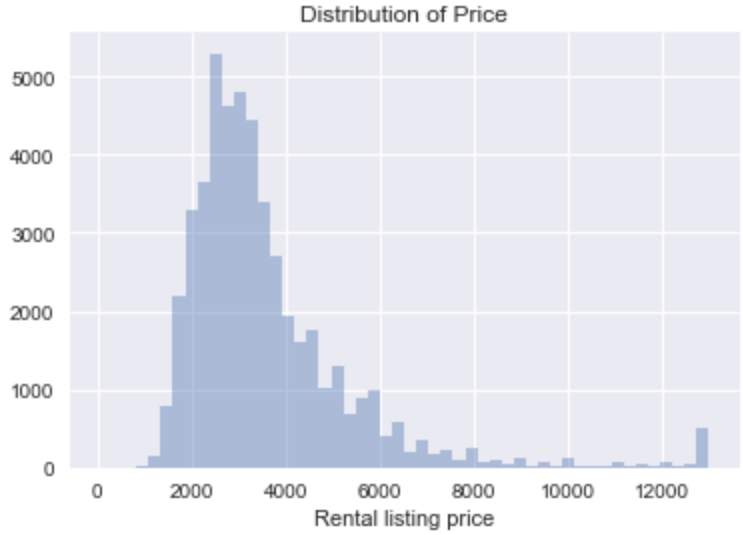
There was no missing or invalid data in the training set and hence we did not have to deal with it.

Outliers

- `Bathrooms:` There was one entry with bathrooms = 10, which is an outlier. Hence removed it.
- `Price:` We have a few entries of price with values above 1000000. These have been removed.

Log transformation

After removing the outliers, we see that the distribution of Price is right skewed. Hence need to a log transform on the price to make the curve look bell shaped.



After log transform of Price, the distribution looks normal.

Feature engineering

I created the following new numeric features from the existing non-numeric features.

1. Num_photos = number of photos for each listing.
   We assume that more number of photos in a listing could generate high interest
2. num_features = number of features for each listing.
   We assume that more number of features in a listing could generate high interest
3. num_description = number of words in the description of each listing.
   We assume that more number of words could generate high interest

Label encoding

We have the following categorical features

"Display_address"
"manager_id"
"Building_id"
"street_address"

I did a label encoding on these features to convert the categorial features to numeric data type.

Implementation

Implementation of label encoding

```
from sklearn import preprocessing
categorical = ["display_address", "manager_id", "building_id", "street_address"]
for f in categorical:
        if train[f].dtype=='object':
            lbl = preprocessing.LabelEncoder()
            lbl.fit(list(train[f].values) + list(test[f].values))
            train[f] = lbl.transform(list(train[f].values))
            test[f] = lbl.transform(list(test[f].values))
            numeric_features.append(f)
```

After label encoding, we have the following numeric features we can use for the models.

```
print (train[numeric_features].head(4))
```

```
      bathrooms  bedrooms  latitude  longitude  listing_id  logprice  \
10            1.5         3   40.7145   -73.9425     7211212  8.006368
10000         1.0         2   40.7947   -73.9667     7150865  8.606119
100004        1.0         1   40.7388   -74.0018     6887163  7.955074
100007        1.0         1   40.7539   -73.9677     6888711  8.094073

        num_photos  num_features  num_description  month_created  day_created  \
10               5             0               95              6           24
10000           11             5                9              6           12
100004           8             4               94              4           17
100007           3             2               80              4           18

        hour_created  display_address  manager_id  building_id  street_address
```

```
10                7          12282         1568         3797          23484
10000            12           9080         1988         8986          23680
100004            3          13719         3733         8889           9827
100007            2          10866          282         1848          14237
```

Benchmarking and Initial Algorithm Testing

Before training, the dataset is split into training (80%) and validation set (20%). The model will be trained from the 80% training data. Different models will be tested using the 20% validation set and the best model will be chosen and fine tuned for lower log loss. In this way, we can avoid overfitting and ensure that the final model generalizes well to unseen data.

The test set provided by renthop will be used for predicting the classes using the best model.

After splitting the data, the models were trained using two supervised classifiers - Logistic regression and Random forest. The log loss was measured for each model with the validation data. The results from the initial model evaluation are as below.

Model	**Log loss**
Benchmark	0.7905
Logistic Regression	0.7838
Random Forest	0.6444

Both the chosen models have performed better than the benchmark with lower log losses. However Random forest has the lowest log loss and hence is the best model for this problem.

Refinement

Each algorithm uses a number of parameters for inputs. Tweaking these parameters can drastically change the performance of the algorithm. Unfortunately, the highest-performing combination of parameters cannot be determined *a priori* but rather must simply be tested. This process is called parameter tuning.

One can imagine each possible combination of parameters as a cell in an n-dimensional grid, where n corresponds to the number of parameters being tuned. This grid is then exhaustively searched for the "best" parameter values. The highest-performing parameter values are then reported, along with their score.

For the random forest model, the following parameters were tuned.

- **N_estimators** : Number of trees in the forest
- **max_depth**: the maximum depth of the tuned tree.
- **Min_samples_split**: minimum number of samples required to split an internal node
- **min_samples_leaf**: The minimum number of samples required to be at a leaf node
- **Class_weight** : Weights associated with class. If not given, all classes are supposed to have weight one.The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data

I fine tuned the model using sklearn GridsearchCV to tweak the parameters to improve its performance.

```
# Using Gridsearch to optimize parameters for Random forest
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
grid_search_params = { 'n_estimators' : [10, 100],
                       'max_depth' : [5, 10, 15],
                       'min_samples_split': (5, 10, 25),
                       "min_samples_leaf": [1, 5, 10],
                       "max_leaf_nodes": [None, 5, 10, 20]
                     }
clf2_opt = RandomForestClassifier(random_state = 50)
gs = GridSearchCV(estimator = clf2_opt, param_grid=grid_search_params)
gs.fit(X_train, y_train)
y_val_pred_clf2_opt = gs.predict_proba(X_val)
print ("Log loss for Random forest model optimized with Grid search:", log_loss(y_val,
y_val_pred_clf2_opt))
print("Best params: ", gs.best_params_)
print("Best score: ", gs.best_score_)

Log loss for Random forest model optimized with Grid search: 0.599600124827
```

The best values for the parameters of the random forest determined from Grid search are as below:

```
Best params:  {'max_depth': 15,
               'max_leaf_nodes': None,
               'min_samples_leaf': 1,
               'min_samples_split': 10,
               'n_estimators': 100}
```

IV. RESULTS

Model Evaluation and Validation

Random forest classifier, optimized with the best parameters has the lowest log loss and is the best model for this problem. This model uses a forest of 100 decision trees, has a maximum depth of 15, and stops splitting once the samples size is less than 10.

This tuned model is used to predict the output classes of the test data and the output probabilities are stored in output.csv file.

```
y_test = gs.predict_proba(X_test)
y_test_prob = pd.DataFrame(data = y_test, columns=gs.best_estimator_.classes_)
y_test_prob.to_csv("output.csv", index=False)
```

Here is a sample of the output predictions of the test data

```
y_test_prob.head(5)
```

	listing_id	high	medium	low
0	7142618	0.077955	0.373362	0.548682
1	7210040	0.086118	0.118656	0.795226
2	7103890	0.042414	0.177330	0.780255
3	7143442	0.104665	0.426603	0.468732
4	6860601	0.048196	0.185749	0.766055

This file was submitted to Kaggle and the submission got a log loss score of 0.61139 for the test data.

Your submission scored 0.61139.

This model generalizes well to unseen data. It's performance on the test data (0.61139) is very similar to the performance on validation data (0.5996)

Justification

After tuning, the optimized model has a lesser log loss than the original model and hence is the best model for this problem. This model has also been used to predict on the test data and has a comparable log loss with the training data.

Model	Log loss

Benchmark	0.769
Random Forest (unoptimized)	0.6444
Random Forest optimized	0.6082

V. CONCLUSION

Free form visualization

An important task when performing supervised learning is determining which features provide the most predictive power. By focusing on the relationship between only a few crucial features and the target label we simplify our understanding of the phenomenon, which is most always a useful thing to do. In the case of this project, that means we wish to identify a small number of features that most strongly predict whether a rental listing has 'High' interest or not.

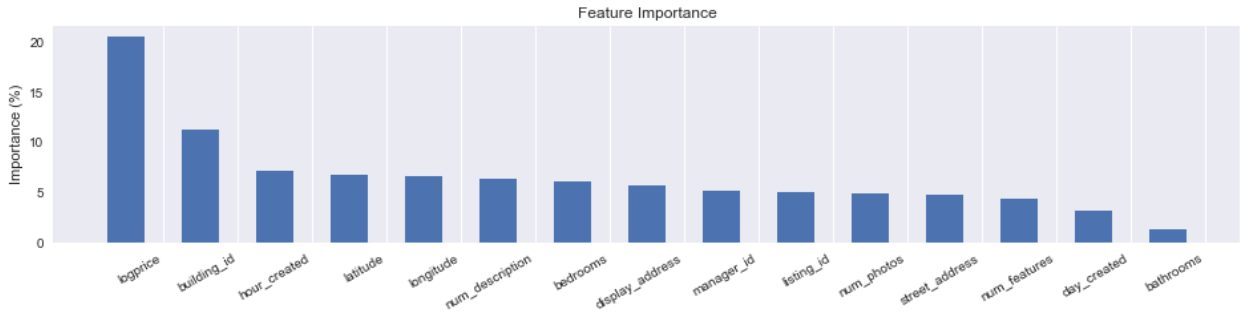
```
# Feature importance
relevance = pd.DataFrame([clf.feature_importances_], index=['importance'], columns = X_train.columns)
relevance = relevance[relevance > 0.01].dropna(axis=1).transpose().sort_values('importance', ascending = False)
display(relevance)

relevance_data = relevance['importance'] * 100
N = len(relevance_data)
index = np.arange(N)
width = 0.5

plt.figure(figsize=(16,3))
plt.bar(index, relevance_data, width)
plt.xticks(index + width, relevance.index, rotation=30, ha='right')
plt.ylabel('Importance (%)')
plt.title('Feature Importance')
plt.grid(which='major', axis='y')
plt.show()
```

	importance
logprice	0.205715
building_id	0.112008
hour_created	0.071414
latitude	0.067211

longitude	0.065693
num_description	0.063752
bedrooms	0.061165
display_address	0.056603
manager_id	0.051518
listing_id	0.050296
num_photos	0.048852
street_address	0.048199
num_features	0.044220
day_created	0.032065
bathrooms	0.013892



The top features that influence the prediction are:

1. Logprice
2. Building_id
3. hour_created
4. Latitude
5. Longitude

Reflection

To recap, following are the steps I undertook in this project:

- Data exploration - Explore the data to get a sense of the input and target features
- Data visualization - Visualize the distribution and relationship between different features
- Preprocessing - Remove outliers from the dataset
- Feature engineering - Create new features from the existing features
- Algorithm selection - Choose 2 classifier algorithms based on the problem at hand and intuition

- Benchmarking - Outline a benchmark against which the chose two classifiers will be measured
- Algorithm - Split the training data into train and validation and train the model using the 2 classifiers
- Metrics and Tuning - Pick the best model that does better than the benchmark. Use gridsearch to tune the parameters for this model
- Testing - Use the optimized model to predict the classes for the test data

One of the challenges for me was to run this project in my local machine. Random forest model takes a longer time to train and using grid search to optimize the parameters takes even longer time. So I couldn't do as many experiments or iterations as I had wanted to.

I was not very pleased to see that the model with the optimal parameters did only slightly better than the unoptimized model.

Also, it was interesting to note the top features picked by the feature_importance. Some of the features I had intuitively considered as important like the num_photos, num_features ranked lower than I had expected. The top features influencing the prediction seem to be the price, building_id, hour_created, geographical location (latitude, longitude).

Improvement

I have not used the images in the dataset in my model. For further improvement, I would consider running the images through some neural network and see if it helps improve the accuracy of the model.

I would also consider using NLP methods like bag of words for the text description feature to extract more information.