Airbnb Rating Prediction System For New Listings

Weicheng Zhu, Yiyi Zhang, Zhengyuan Ding, Zihao Zhao

Source Code: https://github.com/jackzhu727/DS-GA1001-Final-Project

Contents

1 Business Understanding

Nowadays, Airbnb, as an online marketplace for providing hospitality services, has become more and more popular among travelers because of its convenience and credible review system between both hosts and guests. As this market gradually matures, the profit for being a host is also apparent. Many people are willing to be a host on Airbnb, renting out their unoccupied rooms or the entire house. However, compared to experienced hosts who received reviews and ratings, new hosts are facing great challenges because they have no referential info for guests to pick their listing among others.

Therefore, we want to use data science techniques to predict a review score based on only the data of house properties and other host-edited descriptions about the listing. Our project aims to build a system which can predict a listing into one of the three classes (0: Excellent, 1: Average, 2: Need to improve) given listing features and descriptions from host. The reasons why we decide to use three classes are also business-oriented. If we simply do a binary classification (0: good, 1: bad), for those listings that got bad ranking, this may discourage hosts. Therefore, 3 classes would be a more considerable design for business scenarios.

The system will bring two main benefits: 1. For Airbnb new hosts, they can use our system to approximate how popular their listings will be and improve accordingly. 2. For guests, they can consider this estimated review score as a reference and expand their range of choices.

2 Data Understanding

We found the data source from InsideAirbnb website and specifically selected the New York City data for our project. These data are sourced from publicly available information on the Airbnb site as stated by this website. There are 7 data files in total involving information of listings, reviews and NYC neighbourhoods (descriptions shown in **Table 1**).

File name	Description
listings.csv	summary information and metrics for listings
listings_detailed.csv	detailed listings information
reviews.csv	summary review date and listing id (listing id, date)
reviews_detailed.csv	detailed review data for listings (listing id, date, comments)
neighbourhoods.csv	neighbourhood list of NY city
calendar.csv	detailed calendar data for listings (available dates and corresponding price)

Table 1: Data Files

According to our business objective, we focus on predicting rating based on listing information, thus we only select ***listing_detailed.csv*** as our data. In this data file, there are 50041 samples and 96 columns in total, among which we set ***review_scores_rating*** as our target variable and all the others as potential independent features. For those features, we dropped some unrelated or useless columns based on our experience and classified the remaining into 4 types: numerical, categorical, date and text. For instance, numerical data includes ***host_response_rate*** and ***host_listings_count***, text data includes ***description*** and ***neighbourhood_overview***. Details of all the features are attached in the appendix.

3 Feature Engineering

Overall, we have applied binning, non-linear transformations (log function), linear transformations (standardization), domain knowledge based feature extraction and other feature engineering skills during the data preparation procedure. The details are described below.

3.1 Basic Data Wrangling

First, we dropped some unrelated or redundant features as listed in part II. Those features are about irrelevant information like url and id, or represent same info with other features, or have no values. Additionally, features with too many missing values($> 70\%$) are deleted. Next, as the motivation is to help predict the rating of a new listing, we dropped review-related features and the feature *'host_since_elapsed'* to avoid data leakage, since a new listing is not able to get review data.

Besides, we generated some new features. A dummy variable *'host_location_if_local'* is created to show whether the host is from New York according to feature *'host_location'*. Additionally, we created dummy

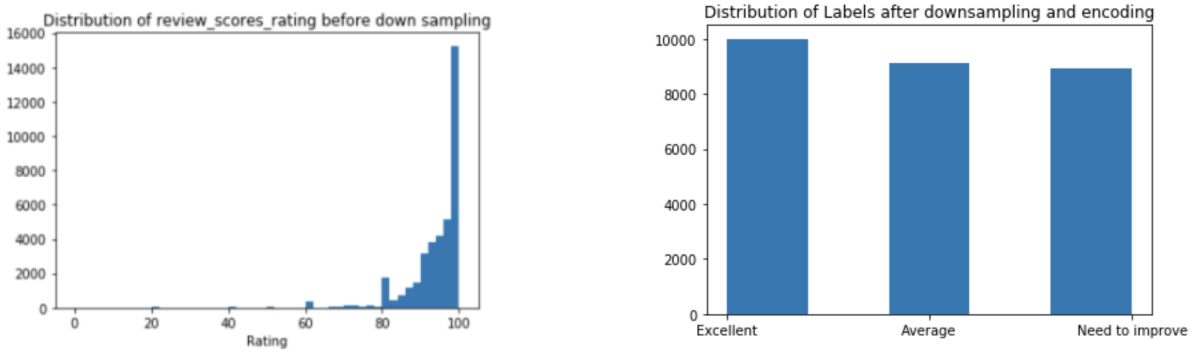
features indicating missing values for text data columns with more than 40% missing values. For example, for *'notes'* column, a dummy column *'has_notes'* is added to our data to show if the host writes notes on his/her listing page.

In terms of categorical features, we split them each into multiple columns of each category and encode them by 0 or 1 dummies. For instance, the feature 'room_type' has 4 categories(*'Private room'*, *'Entire home/apt'*, *'Shared room'*). Then it will be separated into columns like *'room_type_Private_room'*, *'room_type_Entire_home/apt'* and etc. Corresponding columns will be set to be 1 if a data point falls into one of the category and 0 otherwise. Among these categorical features, one of them is ordinal called *'host_response_time'*, which has values *'within an hour'*, *'within a few hours'*, *'within a day'* and *'a few days or more'*. Thus we encode it by 0, 1, 2, 3 respectively to show the order.

3.2 Data Transforming

3.2.1 Non-text Data

For the purpose of predicting a corresponding review class, we set the *'review_score_rating'* as the label (target variable) and encode it into 3 classes by different quantiles: above 50% quantiles = 0, 25%-50% quantiles = 1, below 25% quantiles = 2. Then we drop the rows which their label is missing and downsample the 0 class to balance our dataset. Eventually, we get the numbers of instances in three classes roughly equal.



(a) Distribution of Labels Before Processing

(b) Distribution of Labels After Processing

Figure 1: Data Distribution of Labels

Then we fill missing values in multiple ways: for categorical variables, we replace missing values with zero. For numerical variables, we replace missing values in columns about house properties (e.g.*number of beds*, *number of bathrooms*) with their medians; we replace missing values in columns about host himself (e.g.*host_response_time*) with their means and create a new binary feature indicating the existence of missing value. We introduced missing flags because distribution of these columns are often skew, so means can be highly-biased estimates for missing values.

Then, we check distributions of all numerical variables. We take log of numerical variables with long tails (*'price'*, *'security_deposit'*, *'cleaning_fee'*, *'calendar_updated'*) to reduce skewness. Also, we standardize all numerical variables in this dataset to normalize their means and variances. Finally, we randomly split the dataset into two subsets without overlapping: training dataset (for training and cross-validation) and testing dataset (for checking the final performance on model's predictive abilities).

3.2.2 Text Data

After we finalized the non-text dataset, we think there are few features related to the surrounding or transit info about listings which should be quite important criterion for rankings of Airbnb. Although there is a feature named *review_scores_location*, it is not available for new listings. So we cannot refer to it beforehand. Moreover, it is a review-based component of total rating, which is our target variable, so directly using this column causes data leakage. Hence, we attempt to use NLP techniques to mine location and transit information from textual description written by house owners. This means we will train embedding vectors for words in descriptions as proxies for a review score on location.

review_scores_location is a 0-10 scale grade for guests to evaluate the location. We transfer the grades to 3 classes: above 50% quantiles = 0, 25%-50% quantiles = 1, below 25% quantiles = 2. After extracting 5 text columns that are relevant to location *transit*, *neighborhood_overview*, *summary*, *space*, *description*, we concatenate along columns and tokenize them (using nltk [4] packages to remove punctuation, stop words ,non-letter characters and standardizing into lowercase).

We use high-dimensional embedding vectors to represent words instead of simply conduct feature vectorization with countvectorizer or TfidVectorizer because they can contain more specific information of each words and avoid sparsity problem. We then use Bag-of-Words neural network to train embedding vectors. BoW neural network first looks up words from sentences in word dictionary and maps them into an embedding matrix, then processes the embedding matrix with activation function and one fully connected layer. It outputs a softmax of three categories probabilities and trains the embedding vectors with backward propagation to minimize the loss between outputs and location review scores. To minimize the cross-entropy loss, the algorithm updates the parameters in the embedding matrix in every step with the ADAM(Adaptive Moment Estimation) optimizer. At the end of training, we acquire a trained embedding matrix with trained parameters. Using this matrix, we could then transform text data into an embedded word vector as a proxy for location scores and combine these with non-text data to build our main models in part 5.

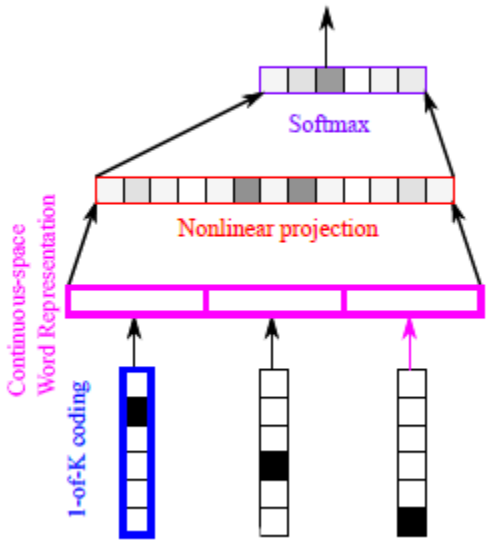
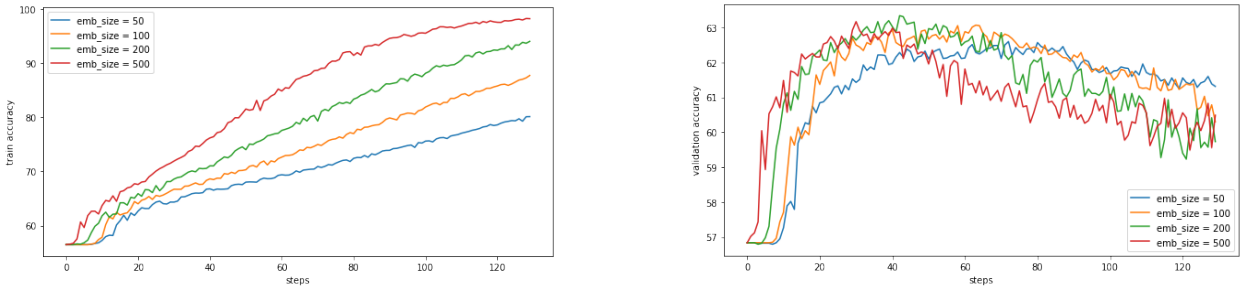


Figure 2: Bag-of-Words Model[6]

Specifically, to build this NLP model, we first split the dataset into a training set (0.75) and a validation set (0.25) (same separation ration in non-textual part of dataset). Then we use the training set to build a vocabulary that contains all unique text tokens and count their frequencies. After sorting the vocabulary

by their frequencies, we assign a unique index to each of the text tokens and set index token 0 for padding index and 1 for unknown words. The correspondence between text tokens and index tokens then allows us to transform every text data into a list of index tokens. Then we build the structure of the model with PyTorch and use backward propagation to train it.

To get a best NLP model, we tune two hyper-parameters: embedding size and learning rate. Initially we have embedding size = 200, and learning rate = 0.001 and the highest validation accuracy is 62.57%. First, embedding size controls the dimension of the embedding matrix and the length of the embedded vector. A too low embedding size cannot compress enough information while a too large embedding size will easily lead to overfitting. Therefore, it is crucial to take an appropriate embedding size. We tuned it among [50, 100, 200, 500]. From the figure 3b we could observe that, although 200 and 500 gives high validation accuracy at the begining, their validation accuracy go down seriously after 60 steps. Compared to these, smaller embedding size gives more stable results along with the training process. Since both the training and validation accuracy for embedding size 50 is too low, we finally choose 100 as the optimal.

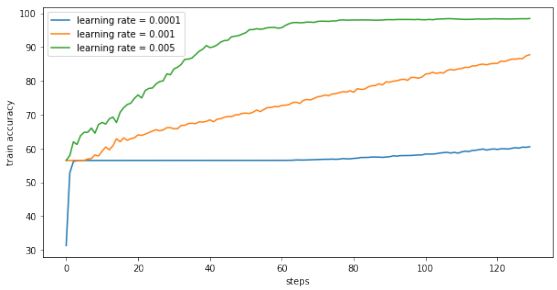


(a) Training Curve

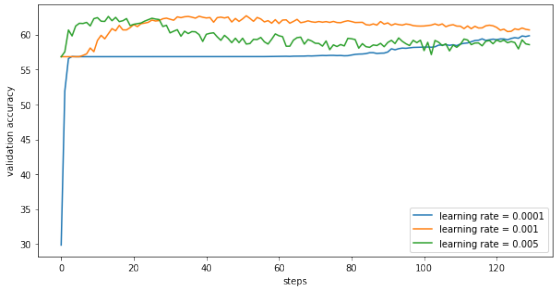
(b) Validation Curve

Figure 3: Accuracy on Multiple Embedding Sizes

Learning rate of the Adam optimizer is the step size in stochastic gradient descent. With a high learning rate, the accuracy would fluctuate very much. With a low learning rate, on the other hand, the training process is slow and it would take a long time to get a high accuracy. We tuned three learning rates: 0.0001, 0.001, 0.005. Obviously from the validation curve 4b, we can conclude that 0.001 is the best choice for learning rate.



(a) Training Curve



(b) Validation Curve

Figure 4: Accuracy on Multiple Learning Rates

Finally, we took embedding size = 100 and learning rate = 0.001, and the validation accuracy is 63.3%.

With this BoW model, we successfully generated the embedding matrix that transforms all the location-related textual description in the dataset in to 100-dimensional vectors. Then we used PCA to reduce the dimension of the vector from 100 to 70, keeping 87.2% variance of the data.

4 Feature Selection

According to Jason Brownlee (2014), feature selection "reduces overfitting, improves accuracy, and reduces training time[1]. As we mentioned before, our finalized dataset contains 28056 rows with 177 non-text columns. There are so many dimensions and some may be highly correlated, thus it would be necessary to do feature selection on our non-text columns.

This project uses techniques involving 3 major feature selection methods: filter methods, wrapper methods and embedded methods as summarized by Jovi, Brki and Bogunovi (2015).[2] To get a reference of the performance of different methods, we built a baseline model with non-text data and the accuracy is 38.5%. We compare the results of different methods and choose the best one for the models in part 5. A summary of all the methods we tried are shown below and the corresponding performances are shown in Table 2.

1. Filter Methods

- Mutual Information: By calculating the mutual information between features and target variable, we rank the feature importance and select top100 features.

2. Wrapper Methods
   - Low Variance: If one column shows a constant value, then it does not help for the model to discriminate different labels. Therefore, we reduced dimensions by dropping off columns that have low variances. On the training data set, we set the cutoff to be 0.1 and maintain 74 columns.
3. Embedded Methods
   - Lasso: Lasso regression adds an L1 regularization on the cost function of linear regression, which prevents overfitting and additionally reduces the coefficients of less important features to zero. Therefore, it can also be used for feature selection. After tuning the regularization coefficient, we picked 0.001 and chose 90 best features based on the performance on the baseline model.
   - Tree-based: The tree-based model can be used to calculate feature importance based on Gini importance. It works in the way that better features are generally closer to the root. We specifically use extra-tree (extremely randomized tree) classifier as the model, which randomly selects a cut-point by K features. This is more efficient in that it alleviates the computational efforts in deciding the optimal cut-point compared to random forests and reduces variance due to the randomness as claimed by Geurts.[3] As regard to the threshold of feature importance, we use the default value of the SelectFromModel in sklearn package, which is 10^{-5}. After tuning the model to optimal, we kept 74 best features and reached an accuracy of 45.1% on our baseline model. And this is our final choice for feature selection based on its performance.

	Baseline Accuracy	Mutual Information	Low Variance	Lasso	Tree-Based
Naive Bayes	38.5%	42.9%	43.9%	44.4%	45.1%

Table 2: Validation Accuracy Comparison

5 Modeling

5.1 Data Mining Algorithms

Speaking of the possible data mining algorithms, as our project is dealing with a classification problem, Naive Bayes would be the first and the simplest algorithm we could try. However, since Naive Bayes makes the assumption that all the features are independent, which is not the general case in our project, we will only refer to it as our baseline model. For other algorithms, here we discuss about Logistic Regression, SVM, Random Forest, Gradient Boosting Classifier and Ada Boost Classifier to see which model would perform the best during cross-validation.

	Pros	Cons
Logistic Regression	• Easy to implement and very efficient to train	• Cannot solve nonlinear problems
Decision Tree	• Easy to interpret and implement • Prediction/scoring is cheap • No feature engineering necessary: • Can accept any types of data	• Easy to overfit, lack of generalizability • Greedy: not very stable, small changes in data can give very different solutions
Random Forest	• Flexible • Reduce variance • Enhance the accuracy of the weak algorithm to a better extent	• Not as easy to visually interpret • Heavier computational resources required
Gradient Boosting Classifier	• Lots of flexibility with the choice of loss functions, adaptable to the characteristics of the studied problems	• Many parameters which can interact and influence heavily the behavior of the approach (number of iterations, regularization parameters, etc.) • Overfitting can occur if values of parameters are not suitable • Computationally intensive
Adaboost Classifier	• Can achieve similar classification results with much less tweaking of parameters or settings	• Cares for accuracy thus only good for balanced dataset • Sensitive to noisy data and outliers

After the analysis of the 5 models shown above and combining our dataset and business goal, we decided to use Logistic Regression, Random Forest and Gradient Boosting Classifier in the following procedure.

5.2 Baseline Model and Evaluation Framework

Naive Bayes models are a group of extremely fast and simple classification algorithms that are often suitable for very high-dimensional datasets. Because they are so fast and have so few tunable parameters, they end up being very useful as a quick-and-dirty baseline for a classification problem. Thus, we choose Naive Bayes as our baseline model with both text and non-text features put in training and it reaches an accuracy of 43.31%. The detailed results are shown in Figure 5, where the diagonal accuracies is not very high showing that the model mixes up three classes often. For evaluation, we use cross-validation accuracy score and confusion matrix to measure the performance of the three models we chosen in the last part. We chose to use logistic-loss as our loss function and apply confusion matrix as our evaluation metric.

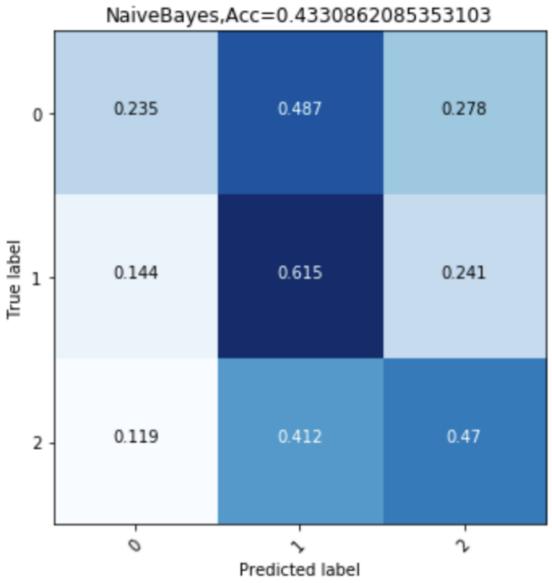


Figure 5

5.3 Hyper Parameters Tuning

Now we conduct hyper-parameter tuning on the three selected models.

- Logistic Regression

  We use cross validation accuracy as a reference for performance measurement. The results are shown in Figure 6 of tuning the strength of regularization C. Note that the fitting graph is reaching a sweet

spot when C=0.01, thus we choose this value as optimal. We also tried tuning on the solver but the performances are similar. Finally, the best model we get reaches an cross-validation accuracy of 49.43%, which is 6.1% higher than the baseline.

Figure 6

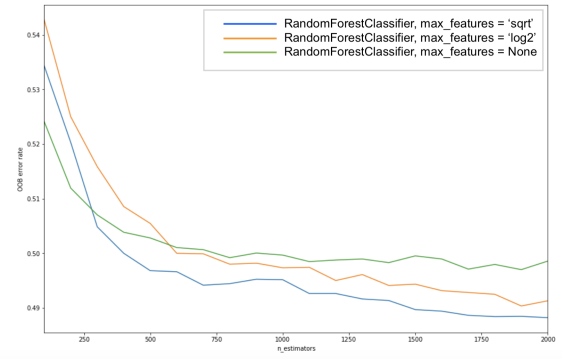
Best Model Non-default hyperparameters: C=0.01, solver='lbfgs', multi_class= 'multinomial'

- Random Forest

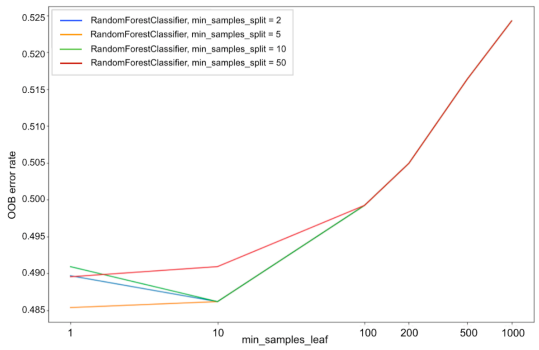
  For random forest, we use out-of-bag error rate(OOB) instead of cross validation for tuning to get the process faster. Firstly we tune on the n_estimators(the number of trees added to the model) and max_features(maximum number of features to consider to decide splits). The results(in Figure 7a) indicates that 1500 n_estimators is probably a suitable one since theres no significant decrease between 1500 to 2000. The square root is best one for max_features as its curve is the lowest.

  Then we tune on the min_samples_leaf and min_samples_split in Figure 7b and chooses 10 for both hyperparameters as its accuracy doesnt differ much from 5 min split and 1 min leaf while it saves computational efforts.

  Best model non-default hyper-parameters: n_estimators = 1500, max_features = 'sqrt, min_samples_leaf = 10, min_samples_split = 10



(a)



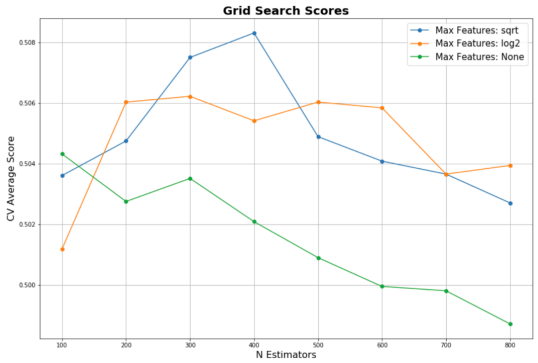
(b)

Figure 7

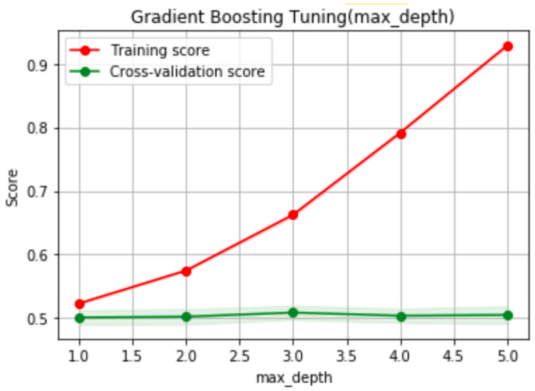
- Gradient Boosting

For tuning of gradient boosting, we use cross-validation to evaluate the performance. Figure shows the tuning process on max_features and n_estimators and its clear that the accuracy score is the highest (50.83%) when n_estimators is 400 and max_features uses square root method.

To reduce the computational burden, we tune on the max_depth and chooses 3 as the best tradeoff between accuracy and training time.(Figure 8a)



(a)

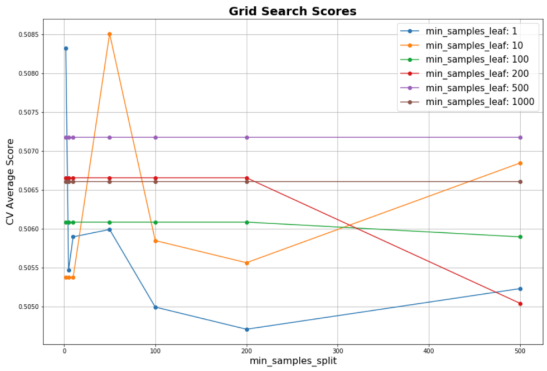


(b)

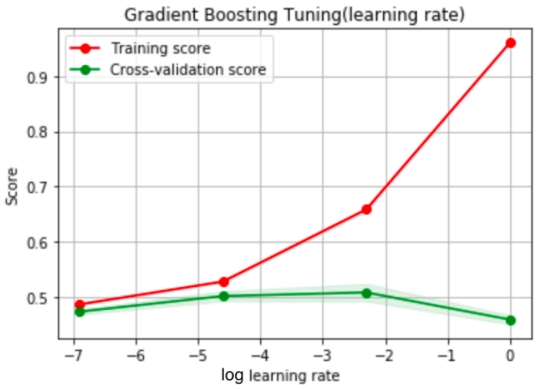
Figure 8

For min_samples_split and min_samples_leaf, we choose 50 and 10 as this combination has the highest accuracy shown in the Figure 8b.

Additionally in Figure 9a we tuned on learning rate and pick 0.1 as it reaches the best accuracy.



(a)



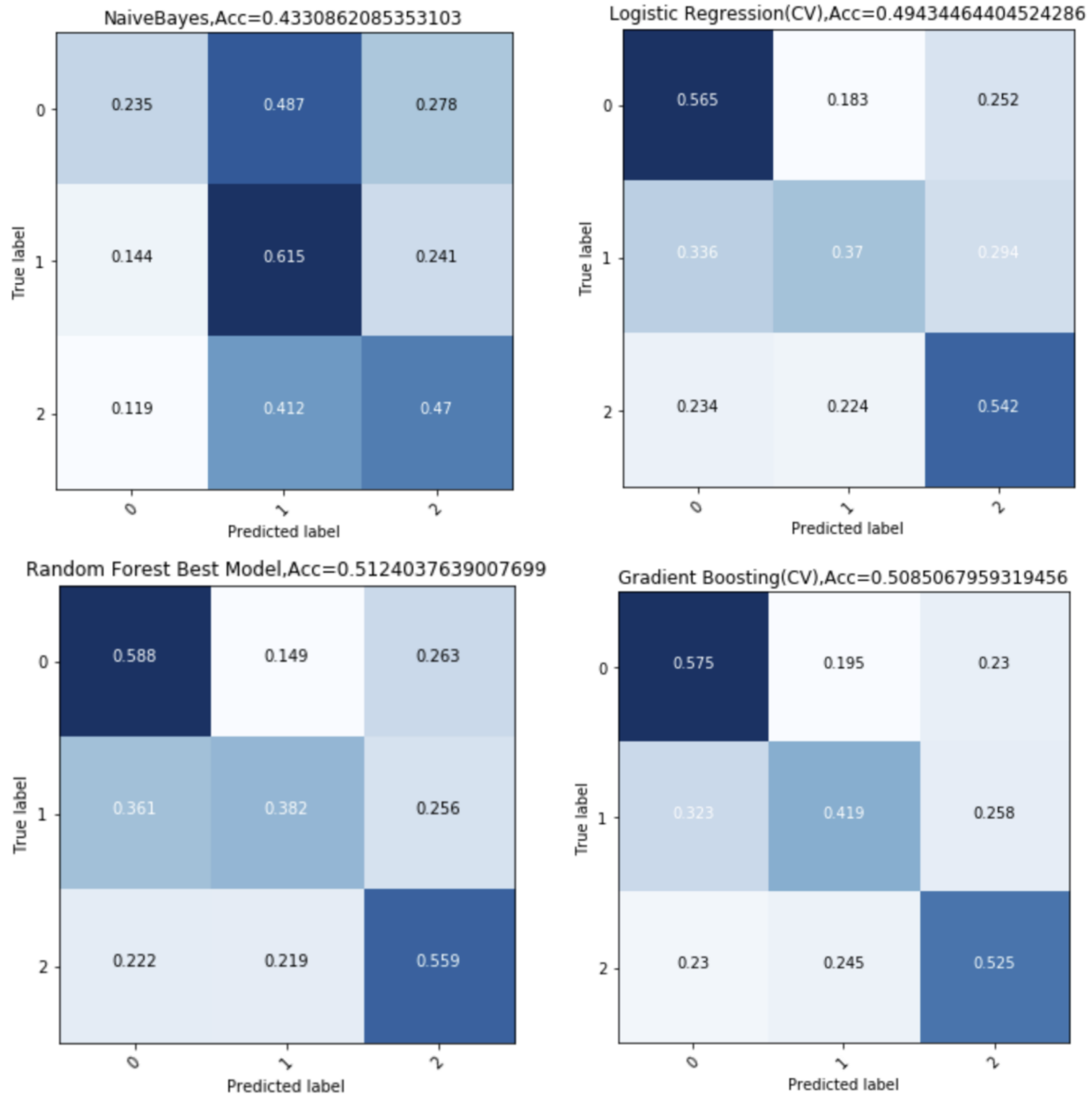
(b)

Figure 9

Best Gradient Boosting non-default hyper-parameters: max_features = 'sqrt', learning_rate=0.1, n_estimators = 400,min_samples_split = 50, min_samples_leaf = 10, max_depth = 3

6 Model Selection

After tuning for all three models and get an optimal one for each, we compared these models and baseline using cross validation as well as confusion matrix to do an analysis of their performances. As we can see in the confusion matrices below, Random Forest displays the highest cross-validation accuracy and the diagonal prediction is comparatively clear. Also, we need to notice the left-bottom corner of the confusion matrices (the predicted labels are 0 while the ground truth is 2), it would cause the most serious business cost in our case. Compared to other models, Random Forest also shows a low false rate at this position. Therefore, we choose to use Random Forest as our final data mining algorithm.



7 Deployment

Now, let's move on to the business deployment scenarios. Thanks to the model parameters and the word embedding matrix, we can build a prediction tool (an online platform) for those hosts who just listed their houses for the first time.

This prediction system may have two business scenarios: First, new hosts can use it to perfect their profile as well as listings. By inputting the properties of their house, house rules and other criterion such

as cleaning fee, security fee deposit, description, transit info, etc., this tool would tell them how good their listing will be. If hosts are not satisfied with the listing, they could then change some policies, add more text info to help them make their listing to get a better class. Second, guests may also refer to our tool and see what might be a completely new listing's problem if it is their last choice. If some of the weaknesses they cannot accept, then they can search for another option instead of hesitating for a while.

We use LIME [5] (local interpretable model-agnostic explanations) to help visualize and interpret our model. As shown in Figure 10, 'host_is_superhost' is the most remarkable feature to distinguish the listings below average to above average. Besides, 'listings with first aid kits' and 'amenities like shampoo' are easier to be classified as excellent or average listings.

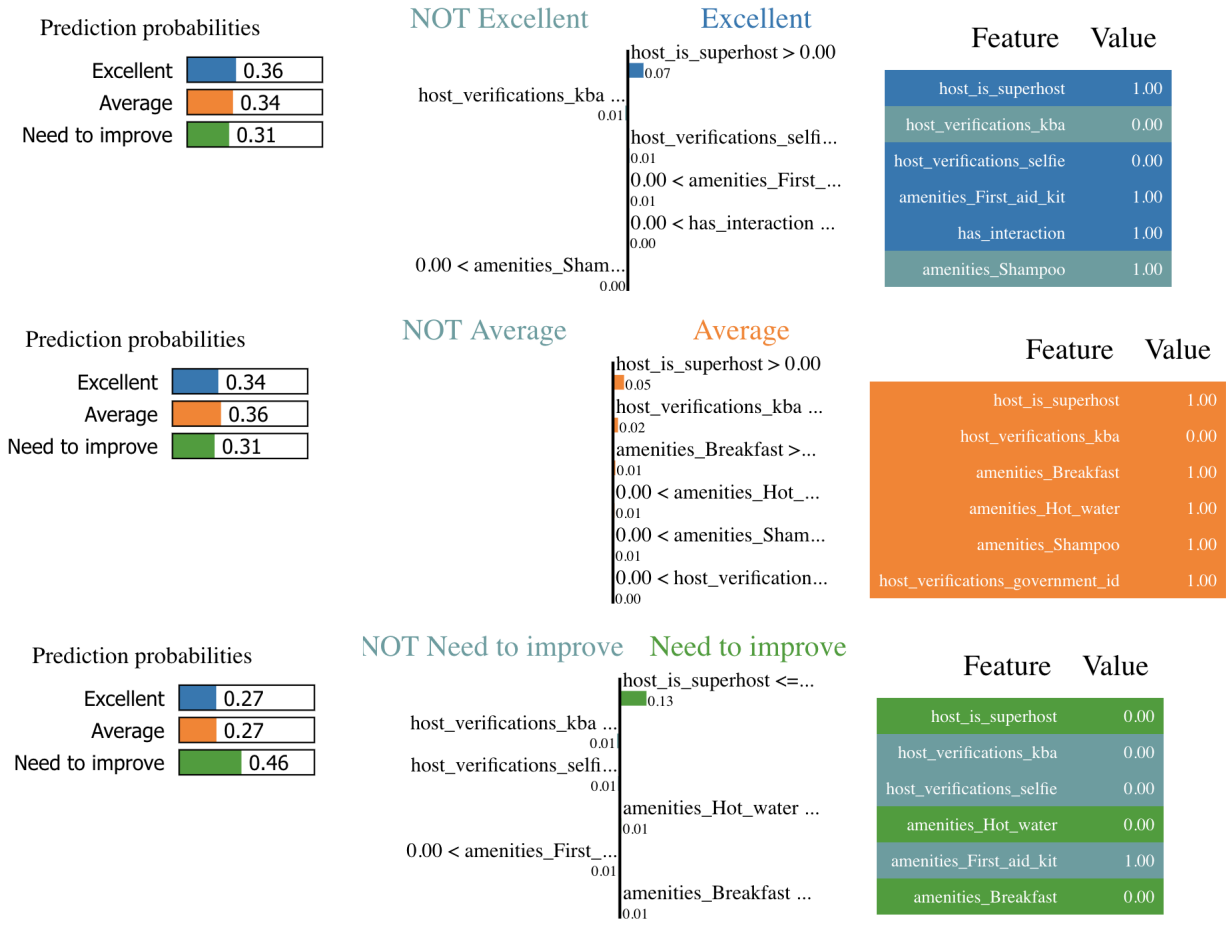


Figure 10: LIME Explanations

8 Conclusion and Future Work

In conclusion, our team tried to build a multinomial classification system to predict a listing's possible review class without knowing any history review info. Our features can be summarized into 3 categories: the properties of the listing, the personalities of the host and the text data about listing's surroundings and location. After several steps we described above, finally, we chose to use Random Forest to build the system. We used tree-based feature selection methods and kept 144 columns as features, then did cross-validation and tuned the hyper parameters. The final test accuracy for 3 classes is around 50.37% (shown in Figure 9).

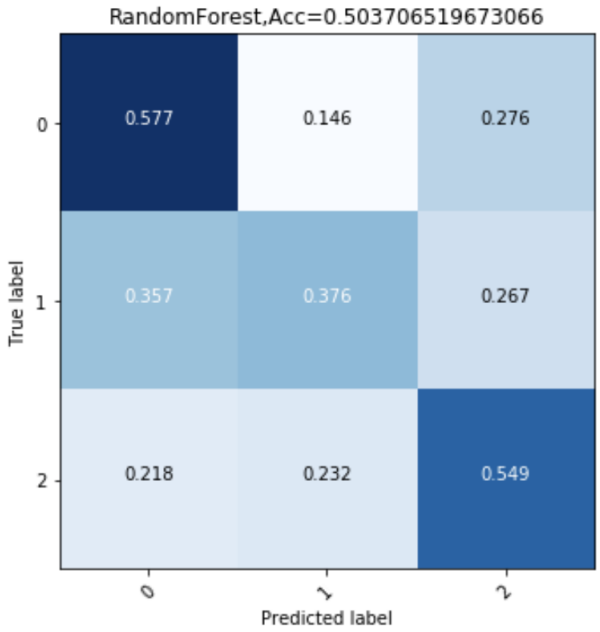


Figure 11

It sounds that the total accuracy is not that satisfactory, but since it is a multinomial classification problem, the base rate would be 33.3% if the model just did a random guess(data balanced in three classes). And our model has improved this rate by nearly 20%. It is a big progress. Since all the data we used are real-world data, we think the model we built has some predictive power and will increase benefits for Airbnb. However, after we retrospected what we learned from lectures, the low accuracy of the model can be explained by the following reasons:

1. Irreducible error: Our model tackles the real-world complicated data and it contains irreducible error. Some close samples may be rated completely different;
2. Customers tend not to rate the average listings: It is common that only impressive listing or bad listings would trigger guests to leave comments, which makes the average listings comparatively hard to differentiate themselves. Not much valuable info can be extracted here.

9 Contribution

1. Weicheng Zhu: NLP Bag-of-Words model, feature engineering, hyper-parameters tuning
2. Yiyi Zhang: non-text data preprocessing, feature selection, hyper-parameters tuning, LIME
3. Zhengyuan Ding: data cleaning, feature engineering, tree-model, hyper-parameters tuning, LIME
4. Zihao Zhao: text data preprocessing, hyper-parameters tuning, research

Appendix A Feature List

- Numerical

  host_response_rate, host_listings_count, host_total_listings_count, accommodates, bathrooms, bedrooms, beds, price, weekly_price, monthly_price, security_deposit, cleaning_fee, guests_included, extra_people, minimum_nights, maximum_nights, calendar_updated, availability_30, calculated_host_listings_count

- Categorical
  - Nominal

    neighbourhood_group_cleansed, property_type, room_type, bed_type, cancellation_policy, host_verifications, amenities
  - Ordinal

    host_response_time
  - Dummies

    host_is_superhost, host_has_profile_pic, host_identity_verified, is_location_exact, has_availability, instant_bookable, is_business_travel_ready, require_guest_profile_picture, require_guest_phone_verification

- Date Features

  host_since

- Text Features

  summary, space, description, neighborhood_overview, notes, transit, access, interaction, house_rules, host_about

- Irrelevant or Useless Features

  id, listing_url, scrape_id, last_scraped, name, experiences_offered (all none), thumbnail_url, medium_url, picture_url, picture_url, xl_picture_url, host_url, host_id, host_name, host_acceptance_rate (no value),

host_thumbnail_url, host_picture_url, square_feet, calendar_last_scraped, requires_license (all false value), host_neighbourhood, market

- Review-Related Features

  review_scores_rating, review_scores_accuracy, review_scores_cleanliness, review_scores_checkin, review_scores_communication, review_scores_location, review_scores_value, reviews_per_month, number_of_reviews, first_review, last_review

- Redundant Features(chosen feature written as the first)
  - availability_30: availability_60, availability_90, availability_365
  - neighbourhood_group_cleansed: street, neighbourhood, neighbourhood_cleansed, city, state, zipcode, smart_location, country_code, country, latitude, longitude
  - calculated_host_listings_count: host_listings_count, host_total_listings_count

References

[1] Jason Brownlee. 2014. Feature selection in python with scikit-learn. https://machinelearningmastery.com/feature-selection-in-python-with-scikit-learn/

[2] Jovi, A., Brki, K., Bogunovi, N. (2015, May). A review of feature selection methods with applications. In Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on (pp. 1200-1205). IEEE.

[3] Geurts, P., Ernst, D., Wehenkel, L. (2006). Extremely randomized trees. Machine learning, 63(1), 3-42.

[4] stevenbird. 2018. NLTK Source. https://github.com/nltk/nltk

[5] marcotcr. 2018. Lime: Explaining the predictions of any machine learning classifier https://github.com/marcotcr/lime

[6] Kyunghyun Cho. 2015. Natural language under-standing with distributed representation.CoRR,abs/1511.07916