Kaggle Competition Writeup

CompSci 671

Kaggle ID: Yuanzhi Lou

1 Exploratory Analysis

1.1 Preprocessing

1.1.1 obviously useless features removal

First, I looked at all the features and initially removed some that were clearly not useful for training, including id, scrape_id, last_scraped, picture_url, host_id, host_name, and name.

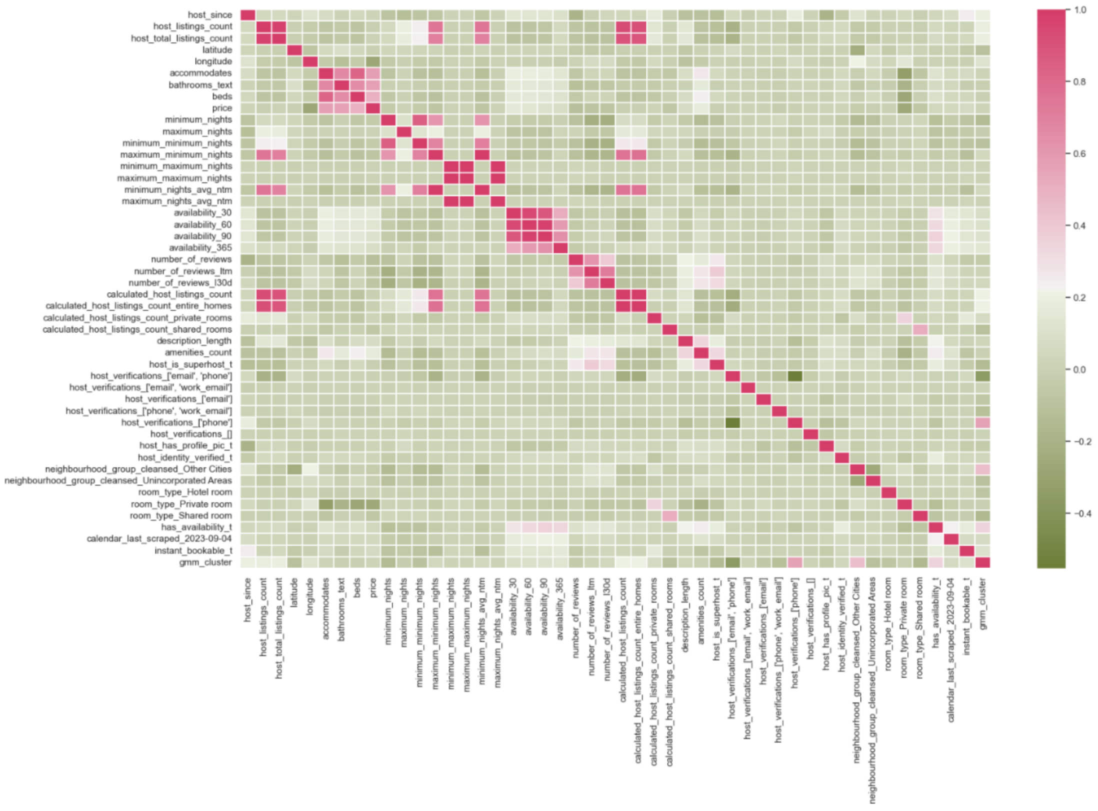
1.1.2 deal with incomplete data

There are four features with incomplete data, including description, host_is_superhost, bathrooms_text, and beds.
The incomplete data in description are filled with an empty string, with a length of 0, as in my algorithm only the length of description is used.
The incomplete data in host_is_superhost and bathrooms_text are filled with the mode, and the one in beds are filled with median.

1.1.3 Digitization

For description and amenities, the length of them is used; For host_since, the date is extracted; For bathrooms_text, numbers inside are extracted; Other non-numerical features are one-hot encoded (except for property_type and neighbourhood_cleansed, they had too many values, so they were deleted).

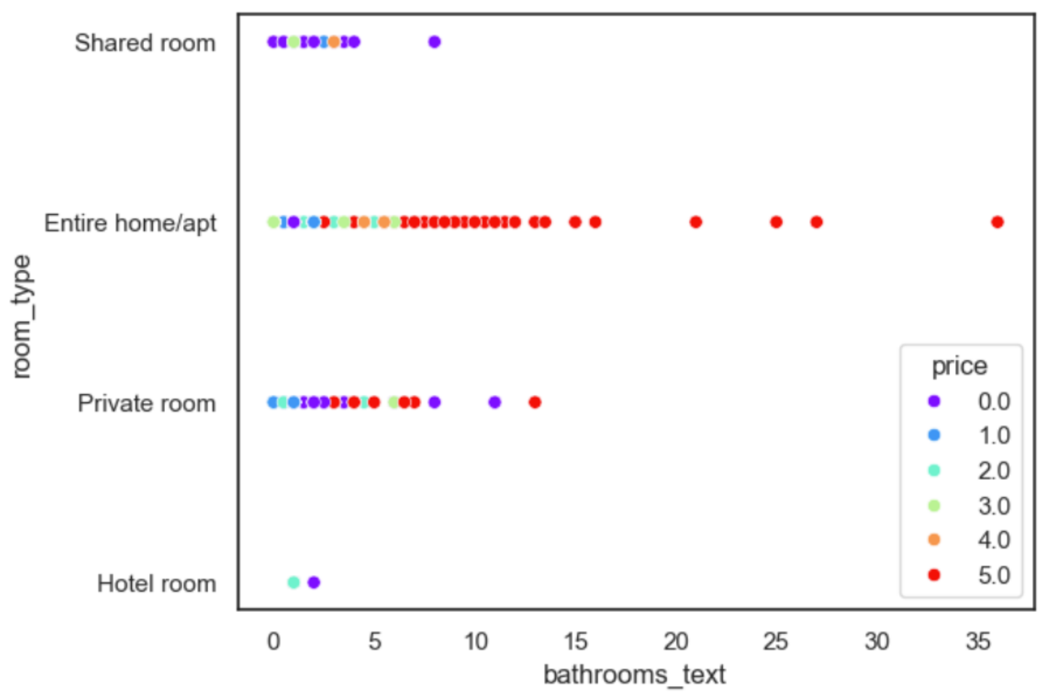
1.2 Correlation



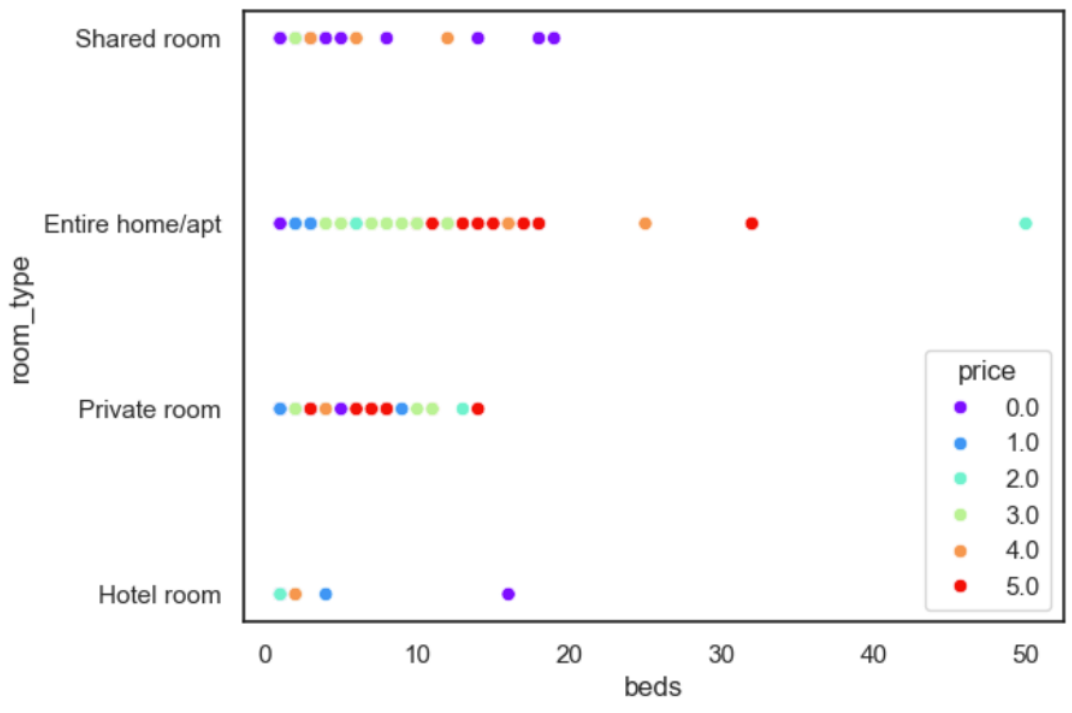
The figure above shows that accommodates, bathrooms_text and beds have strong positive correlation with price, while longitude and room_type have strong negative correlation with price.

1.3 Accommodates, bathrooms_text, beds, room_type.

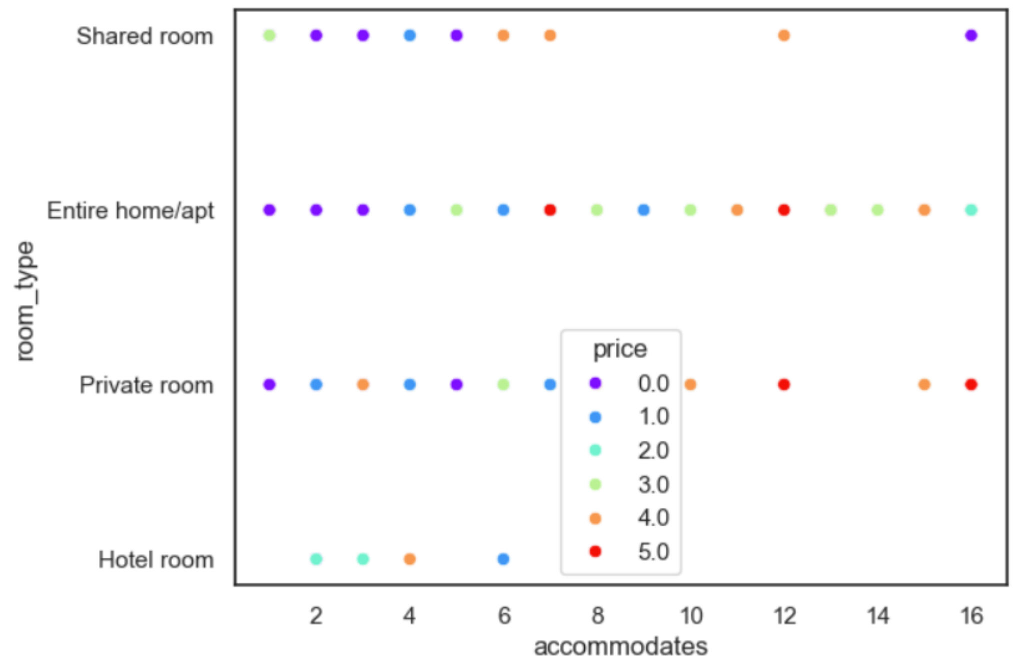
1.3.1 Relationship between bathrooms_text, room_type and price



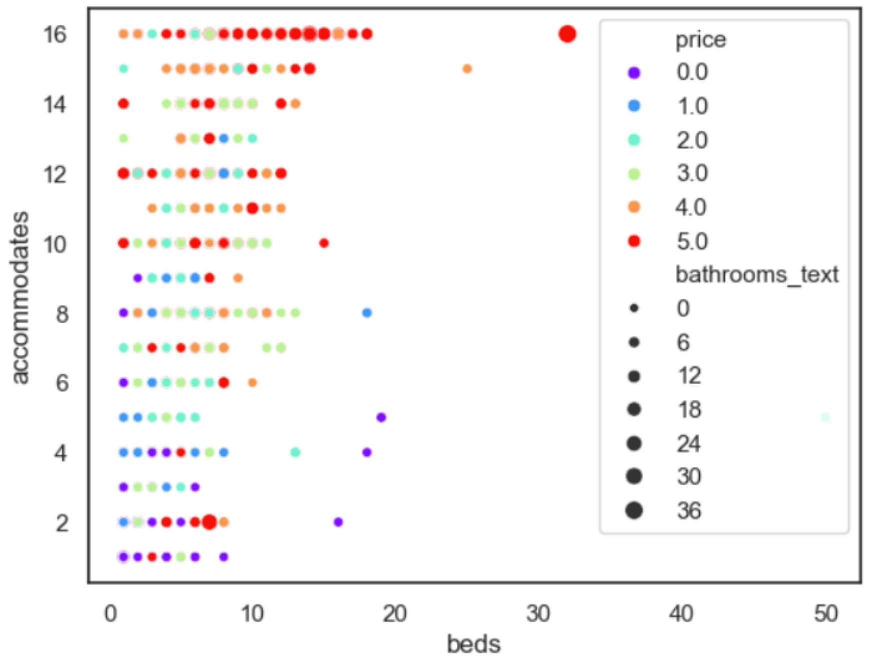
1.3.2 Relationship between beds, room_type and price



1.3.3 Relationship between accommodates, room_type and price.

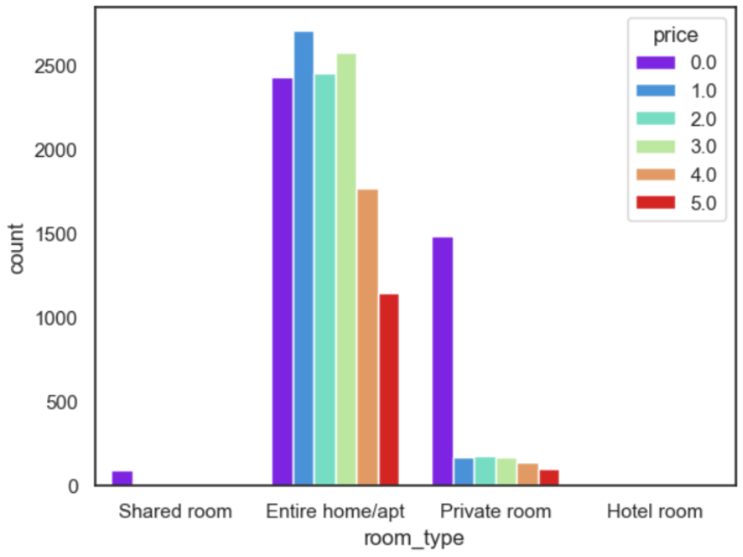


1.3.4 Relationship of accommodates, beds, bathrooms_text and price.



The figures above shows that the one with higher number of accommodates, beds and bathrooms tends to have higher price.

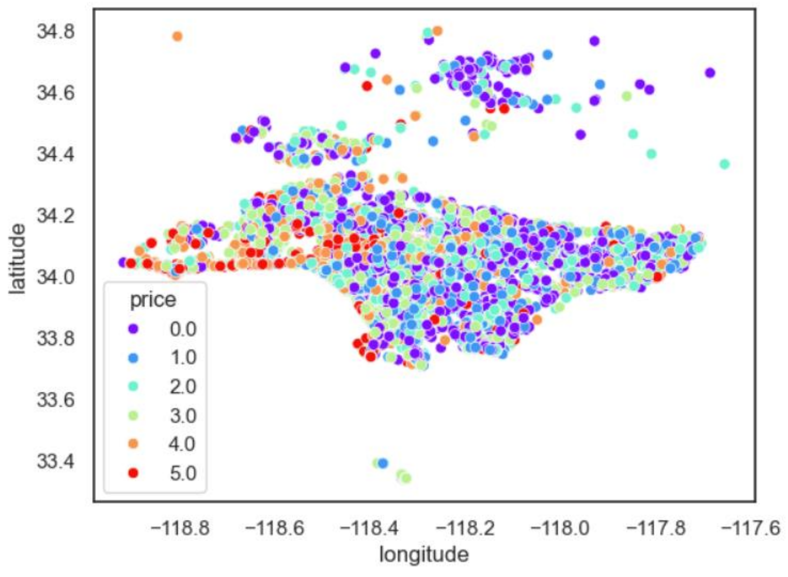
1.3.5 Count of room_type



The figure above shows that shared room and Hotel room have little counts, which means they can be ignored after one-hot encoding room_type.

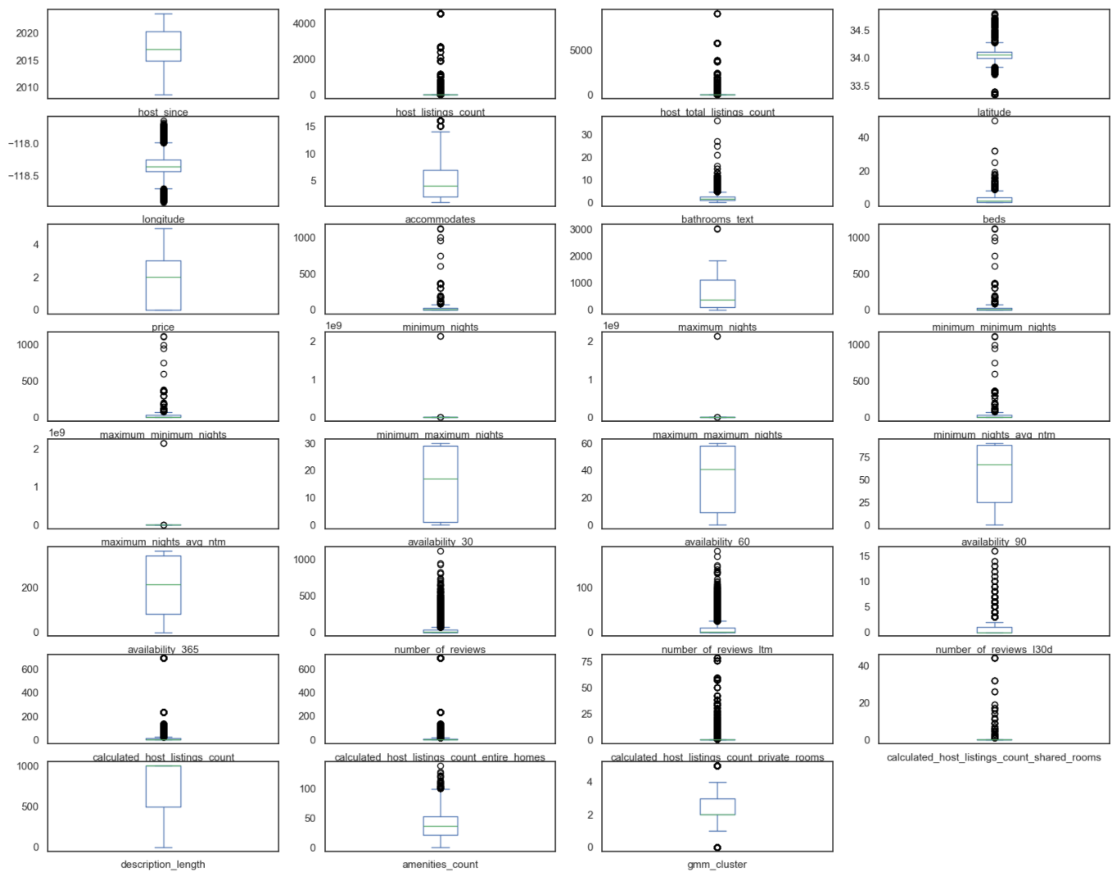
1.4 Longitude and latitude

1.4.1 Relationship of longitude, latitude, and price



The figure above shows that when latitude is in the range from 34 to 34.2 and longitude is in the range from -118.8 to -118.4, the price tends to be higher.

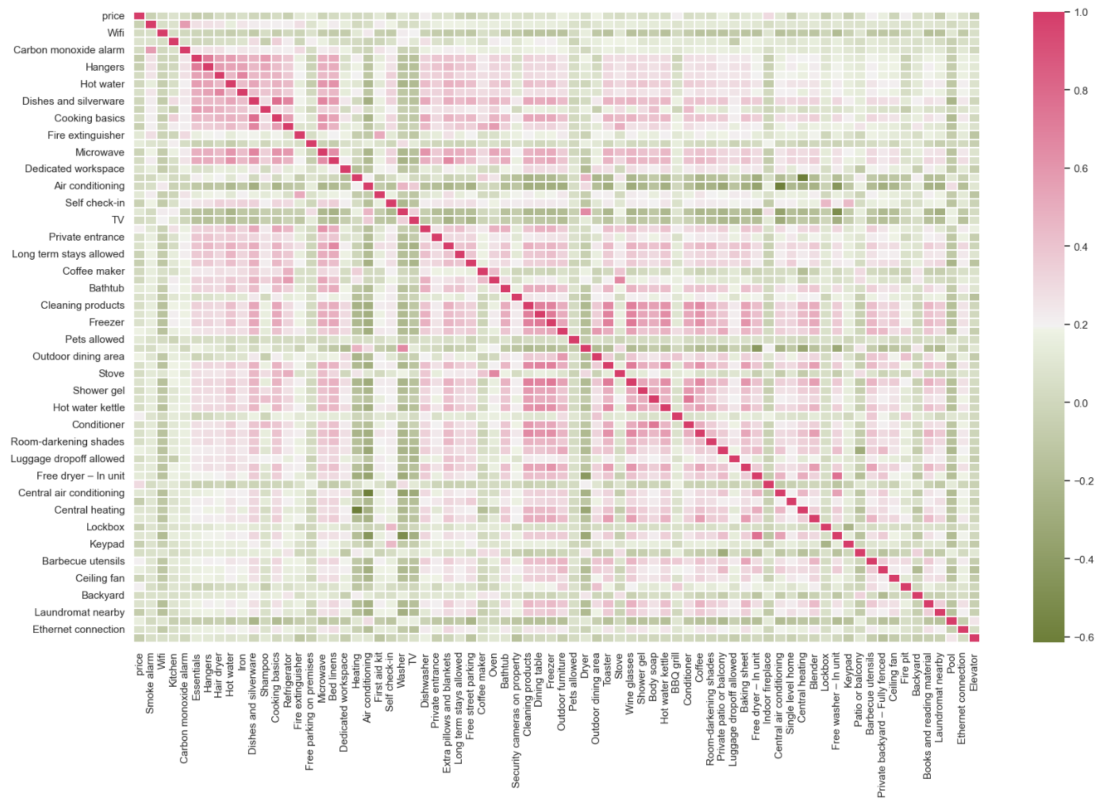
1.5 View Distribution of Elements in Each Feature



The figure above shows that there are outliers in minimum_maximum_nights, maximum_maximum_nights, and maximum_nights_avg_ntm.

1.6 Further analysis of amenities

All amenities with more than 1,000 occurrences are selected and one-hot encoded. Then the correlations between these selected amenities and price are calculated and shown as below.



The amenities with large correlation with price are shown below. These amenities are treated as additional features for training.

```
                                  correlation
Indoor fireplace                     0.294056
Dishwasher                           0.215500
BBQ grill                            0.210879
Barbecue utensils                    0.204971
Fire pit                             0.199987
Outdoor dining area                  0.183014
Sun loungers                         0.179907
Outdoor furniture                    0.179424
Private patio or balcony             0.168137
Private backyard – Fully fenced      0.156681
Pool                                 0.151827
Lock on bedroom door                -0.263517
```

2 Data Splits

In Hyperparameter Selection part, the GridSearchCV method is used to achieve cross-validation. The training data is split into 5 folds, one for validation and the remaining for training. The model is trained on four training folds and then evaluated on the validation fold. This process is repeated for five times and each fold will be treated as validation fold in turn. This cross-validation approach helps in preventing overfitting by ensuring that the model's performance is tested on different subsets of the training data, rather than just on the data it was trained on.

In Train and Predictive Accuracy part, the method train_test_split is used. 80% train data are used for training and 20% train data are used to evaluate the performance of the model.

3 Models

3.1 Random Forest

The first model I used is Random Forest. The RandomForestClassifier in package sklearn is used. In addition, the SMOTE method in package imblearn is used to deal with imbalance of train data (although RF already can deal with imbalance).

The first motivation for choosing RF is that RF has fewer hyperparameters compared to many other algorithms. Therefore, RF is easier to use and tune. The second motivation is that RF can handle many features and is indifferent to the scale of features. This means feature selection and normalization is unnecessary for RF, which simplifies the data processing. The third motivation is that it can provide feature importance, which helps me better understanding features. The last motivation is the high computational efficiency of RF, compared to xgboost and stacking (which will be mentioned later).

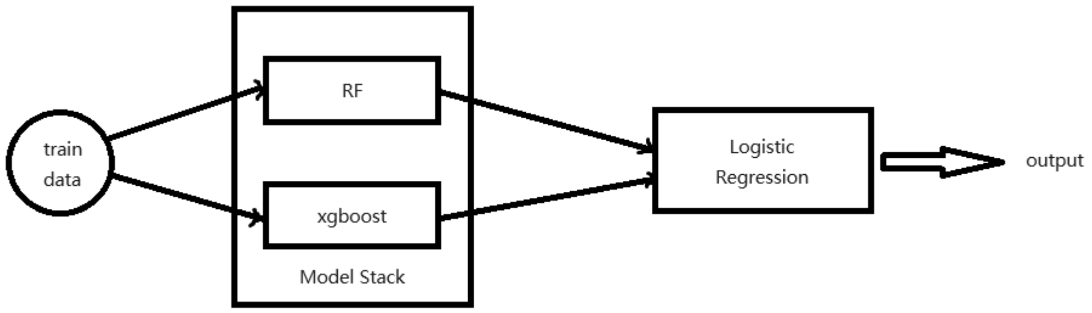
3.2 xgboost

The second model I used is xgboost. The XGBClassifier in package xgboost is used. Xgboost is a distributed gradient boosting library that has been built for maximum efficiency, versatility, and portability. It uses the Gradient Boosting framework to implement machine learning algorithms.

The first motivation of choosing xgboost is that xgboost is known for its high performance and accuracy in many Machine Learning tasks. The second motivation is that xgboost has high-quality library online and good community support, which makes it easy to use. The third motivation is that it has more hyperparameters that can be tuned than RF, which means more room for optimization of the model's performance.

3.3 Stacking

The third model I used is stacking. The StackingClassifier in package sklearn is used. The RF and xgboost mentioned before are used as base estimators which will be stacked together, and the LogisticRegression will be used as the final_estimator which will be used to combine RF and xgboost. The model structure is shown as below.



The first motivation of choosing stacking is its better predictive performance. Stacking can combine the strengths and mitigates the weaknesses of the individual models, leading to a more robust and generalizable model. The second motivation is the ease of implementation, as sklearn provides high-quality implementation of Stacking. The third motivation is that stacking can further reduce overfitting.

4 Hyperparameter Selection

Only hyperparameters in RF and xgboost are tuned. The stacking just used the RF and xgboost with their optimal parameters and default LogisticRegression.

For both RF and xgboost, the method GridSearchCV is used for tuning. In GridSearchCV, the cross-validation is achieved, and the training data is split into 5 folds, one for validation and the remaining for training. In addition, the strategy to evaluate the performance of the cross-validated model is f1_macro.

To save time, I chose to adjust only one or two hyperparameters at a time. This can optimize the model to some extent, although it may fall into local optima.

4.1 Hyperparameter Tuning for RF

In RF, hyperparameters including n_estimators, max_depth, min_samples_leaf, min_samples_split, and class_weight are tuned.

4.1.1 n_estimators

```
'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
```

The best n_estimators is 900.

4.1.2 max_depth

```
'n_estimators': [900],
'max_depth': [5, 10, 15, 20, 25, 30, None]
```

The best max_depth is None.

4.1.3 min_samples_leaf

```
'n_estimators': [900],
'max_depth': [None],
'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

The best min_samples_leaf is 1.

4.1.4 min_samples_split

```
'n_estimators': [900],
'max_depth': [None],
'min_samples_leaf': [1],
'min_samples_split': [2, 3, 4, 5, 6, 7, 8, 9, 10]
```

The best min_samples_split is 2.

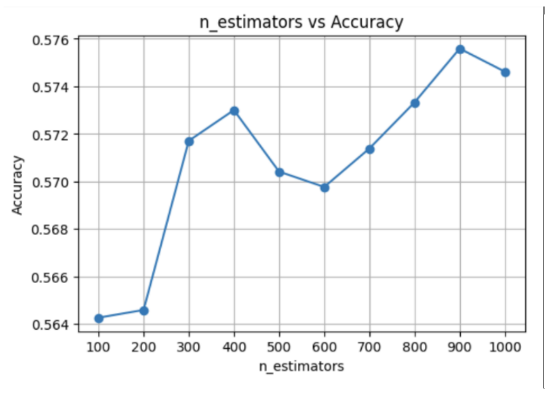
4.1.5 class_weight

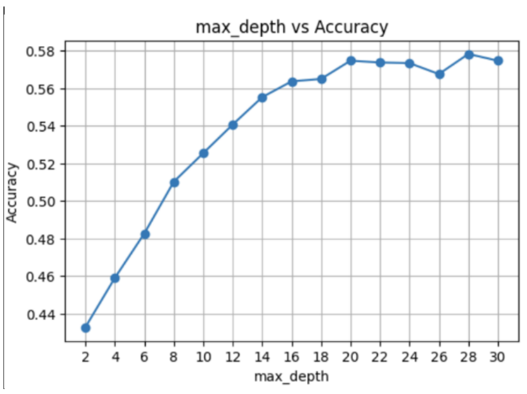
```
'n_estimators': [900],
'max_depth': [None],
'min_samples_leaf': [1],
'min_samples_split': [2],
'class_weight': ['balanced', 'balanced_subsample', None]
```

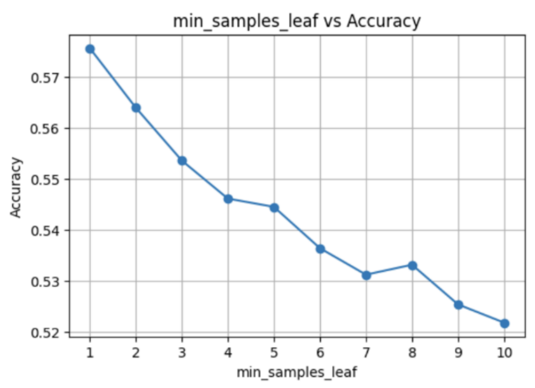
The best class_weight is balanced_subsample.

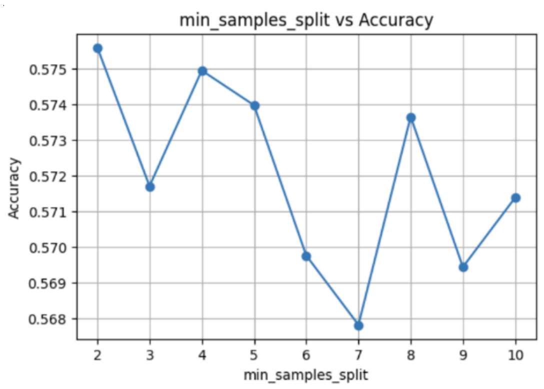
4.2 Hyperparameter vs Accuracy for RF

In this section, the relationships between accuracy and each hyperparameter of RF mentioned above (except class_weight) are shown below.









4.3 Hyperparameter Tuning for Xgboost

In xgboost, hyperparameters including max_depth, n_estimators, learning_rate, min_child_weight, subsample, gamma, colsample_bylevel and colsample_bytree are tuned.

4.3.1 max_depth

```
 'max_depth': [10, 20, None]
```

The best max_depth is 10.

4.3.2 n_estimators and learning_rate

```
'max_depth': [10],
'n_estimators': [300, 500, 800],
'learning_rate': [0.05, 0.3, 1]
```

The best n_estimators is 800 and the best learning_rate is 0.05.

4.3.3 min_child_weight and subsample

```
'max_depth': [10],
'n_estimators': [800],
'learning_rate': [0.05],
'min_child_weight': [0.4, 0.6, 1],
'subsample': [0.2, 0.4, 0.6, 0.8, 1]
```

The best min_child_weight is 0.4 and the best subsample is 0.6.

4.3.4 gamma

```
'max_depth': [10],
'n_estimators': [800],
'learning_rate': [0.05],
'min_child_weight': [0.4],
'subsample': [0.6],
'gamma': [0, 0.2, 0.4, 0.6, 0.8]
```

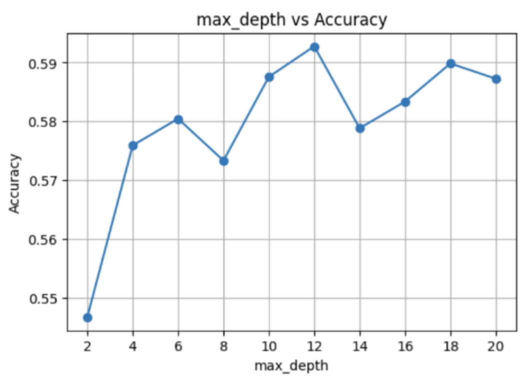
The best gamma is 0.2.

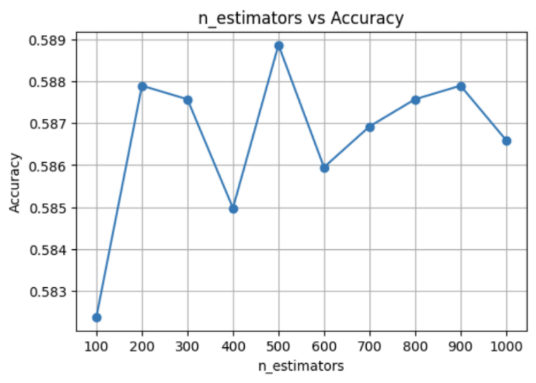
4.3.5 colsample_bylevel and colsample_bytree

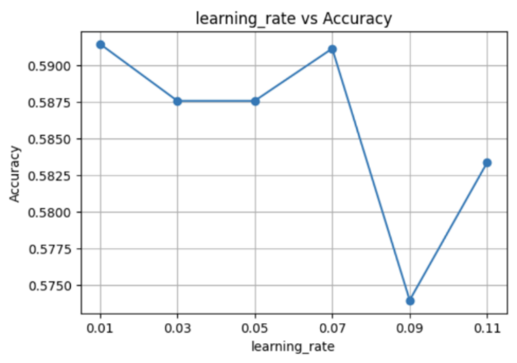
```
'max_depth': [10],
'n_estimators': [800],
'learning_rate': [0.05],
'min_child_weight': [0.4],
'subsample': [0.6],
'gamma': [0.2],
'colsample_bylevel': [0.4, 0.6, 0.8],
'colsample_bytree': [0.4, 0.6, 0.8]
```

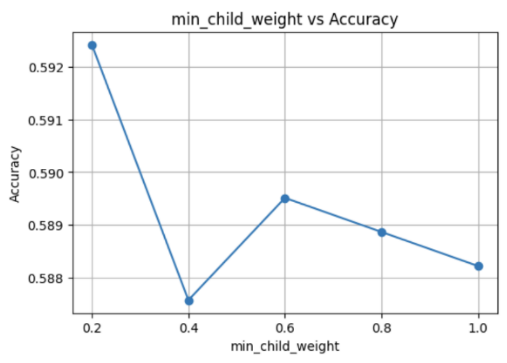
The best colsample_bylevel is 0.8 and the best colsample_bytree is 0.6.

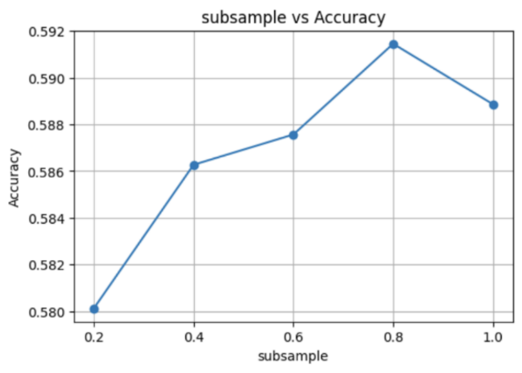
4.4 Hyperparameter vs Accuracy for Xgboost

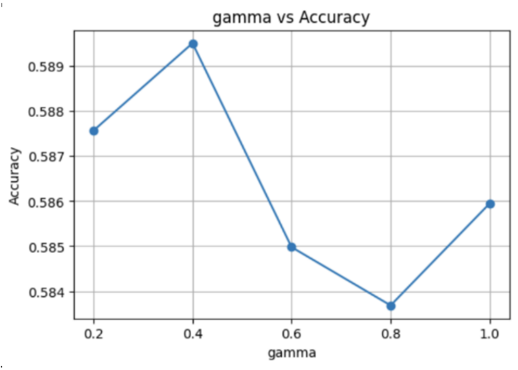


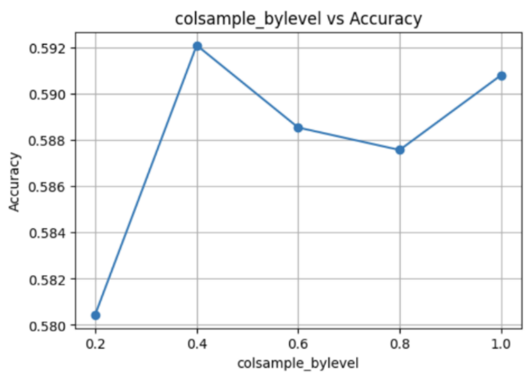


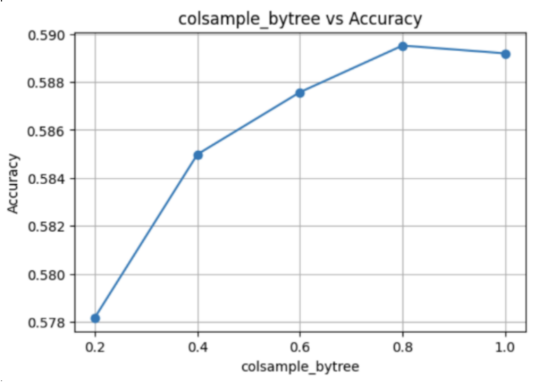












5 Training

5.1 RF

Firstly, for each tree in the forest, a bootstrap subset is drawn from the training data. Secondly, a random subset of features is selected for each tree. Thirdly, in each node of each tree, the best split is selected from the subset of features based on a criterion (here is Gini impurity). Finally, when all trees are built, the forest will aggregate their predictions, and the class with the most vote is chosen as the final prediction of RF.

The wall time required to train RF is 11.1s.

5.2 Xgboost

Xgboost employs a gradient boosting framework for training. Firstly, the algorithm starts with an initial prediction, which is treated as a base model. This base model is then used to calculate the residuals. In each subsequent iteration, the algorithm constructs new trees to predict these residuals, effectively improving the model step by step. In each step, the gradient descent algorithm is used to minimize the loss function and the regularization is applied to avoid overfitting.

The wall time required to train xgboost is 25.2s.

5.3 Stacking

The Stacking is trained using a two-layer structure. In the first layer, base models (here is RF and xgboost) are trained independently on the training dataset with cross-validation. Then, each base model makes predictions on a part of the training dataset that is not trained on. These predictions are treated as features for the second layer. In the second layer, the meta model (here is LogisticRegression) is trained on these predictions to make a final prediction.

The wall time required to train Stacking is 136.2s.

6 Errors and Mistakes

The hardest part of this competition is that when using GridSearchCV to tune hyperparameters, it always costs hours, and I have to re-tune the hyperparameters after features change. This greatly reduces my efficiency.

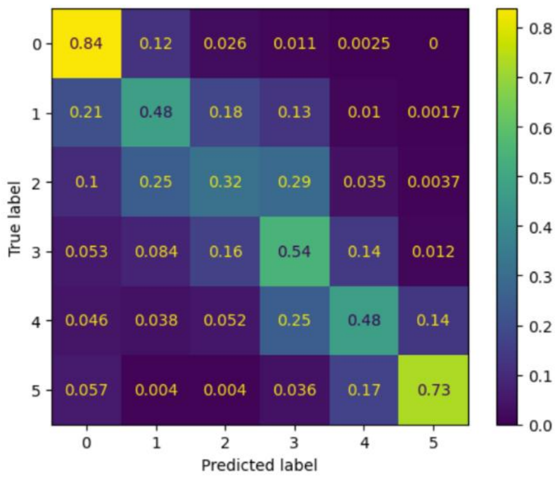
7 Predictive Accuracy

Kaggle username: Yuanzhi Lou

7.1 RF

The train accuracy is 1, and the test accuracy is 0.576.

The confusion matrix is shown below:



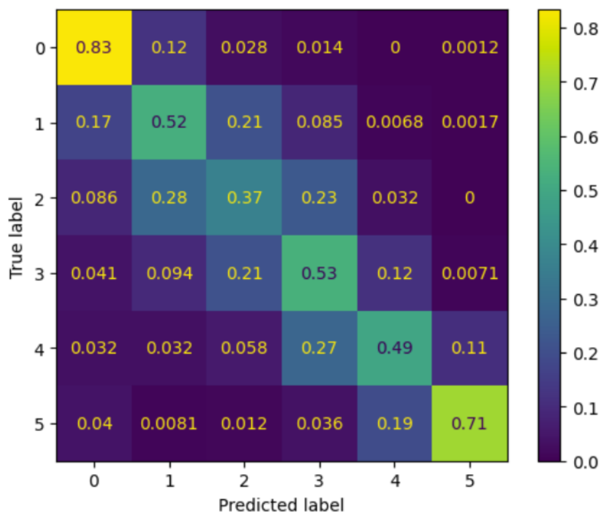
The precision, recall, f1 score and accuracy are shown below:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.84      0.79       810
           1       0.49      0.48      0.48       589
           2       0.42      0.32      0.36       536
           3       0.48      0.54      0.51       561
           4       0.53      0.48      0.50       346
           5       0.76      0.73      0.74       247

    accuracy                           0.58      3089
   macro avg       0.57      0.56      0.56      3089
weighted avg       0.57      0.58      0.57      3089
```

7.2 Xgboost

The train accuracy is 1, and the test accuracy is 0.591.

The confusion matrix is shown below:



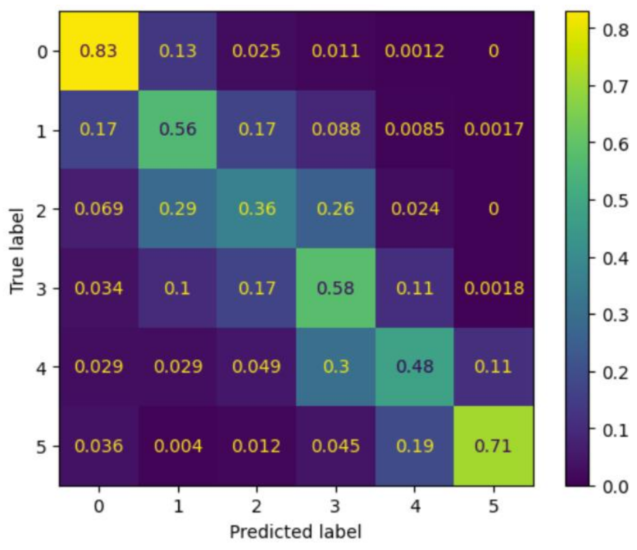
The precision, recall, f1 score and accuracy are shown below:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.83      0.81       810
           1       0.49      0.52      0.51       589
           2       0.41      0.37      0.39       536
           3       0.51      0.53      0.52       561
           4       0.56      0.49      0.53       346
           5       0.80      0.71      0.75       247

    accuracy                           0.59      3089
   macro avg       0.59      0.58      0.58      3089
weighted avg       0.59      0.59      0.59      3089
```

7.3 Stacking

The train accuracy is 1, and the test accuracy is 0.603.

The confusion matrix is shown below:



The precision, recall, f1 score and accuracy are shown below:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.83      0.81       810
           1       0.50      0.56      0.53       589
           2       0.45      0.36      0.40       536
           3       0.51      0.58      0.54       561
           4       0.56      0.48      0.52       346
           5       0.81      0.71      0.76       247

    accuracy                           0.60      3089
   macro avg       0.60      0.59      0.59      3089
weighted avg       0.60      0.60      0.60      3089
```