CMPT459 Milestone 2

Part 1: Decision Tree(Jinze Wu)

1. Which features did you select for your classifiers? Please comment on the reason for your feature selection. If you choose to work on the bonus question, you can add your features extracted from external datasets at this step. (5 points)

We could divide the selected features into 3 parts:

- **Original Data:** 'bathrooms', 'bedrooms', 'created', 'latitude', 'listing_id', 'longitude', 'price'
- **Transformed Data:** 'diff_rooms', 'manager_id_num', 'building_id_num', 'photo_num', 'price_per_bedroom', 'price_per_bathroom',
- **Text Data:** features' one hot attributes(manually selected), descriptions' one hot attributes(manually selected)

Comment:

We used three functions to evaluate the features, including *ExtraTree, SelectKBest-F-score, SelectKBest-Mutual-Information*. We found that *SelectKBest-MI* performed best.

For most of the data, they are numerical, and we found that they are closely related to the interest_level when we did EDA.

For *manager_id_num,* we thought managers were closely related to interest_level and we can not classify manager_id(string) directly, so we used mapping to transformed them to number. For

For *text data*, we did TFIDF and manual selection before, In the initial attempt of classification, we first select the data obtained by simple manual selection.

For the bonus, we create the distance of the house to the nearest subway station.

2. What Python or R libraries did you use for your classifiers? (5 points)

Python Library: sklearn

3. How did you perform cross-validation? Please describe the procedure. (10 points)

In order to make the result more accurate and reliable, we performed cross-validation in two ways.

First method: we use the function cross_val_score() to perform cross-validation. The parameters of the function is "cv=10", i.e. 10-fold cross-validation. We apply this methodology to evaluate the performance of classifier.

Second method: we did 10 fold cross-validation manually. We used the function KFold() to split the index of train data into two parts(1/10 for validation dataset, 9/10 for training dataset). Then, we used the training dataset to learn a model and the validation dataset to get the score as an evaluation of the model. Both of these two methods act similarly in the evaluation process.

4. What performance did the first version of your classifiers achieve on the validation dataset (in cross-validation) and on the test dataset? Please comment on the performance of the classifier. (15 points: 5 points for performance, and 10 points for comments).

Decision Tree(No Parameter)	**Decision Tree(With Some Parameter)**
Validation: The average score(10 Fold) = 0.6504	Validation: The average score(10 Fold) = 0.7121
Test: The score on the kaggle = 19.40	Test: The score on the kaggle = 0.8449

Comment: We thought the performance of the classifier on the validation dataset was considerable, because we didn't tune the parameters of the model. However, the performance of the test data on kaggle was not terrible because the output probability of decision tree was only 1 and 0. We found that once we added the parameter to constraint the max leaves of the tree and the min leaf support, the result will be significantly better. We thought the reason was that if we did not add parameter, the decision tree classifier was overfitting.

5. What actions did you take in order to improve your classifiers? You can modify your dataset or the parameters of your classifier. Please record your modifications in your report. (30 points: 10 points for each improvement)

- **Adding new feature: distance to city centre of new york, 20 most important TFIDF of features and descriptions(10 features, 10 descriptions)**

Before I added new data, I tried to adjust parameters several times and used cross validation and kaggle to evaluate the performance of the classifier. I found that it has been upgraded to the best, but I think the result can be better, so I added some new data. Besides, I deleted the munually selected text data.

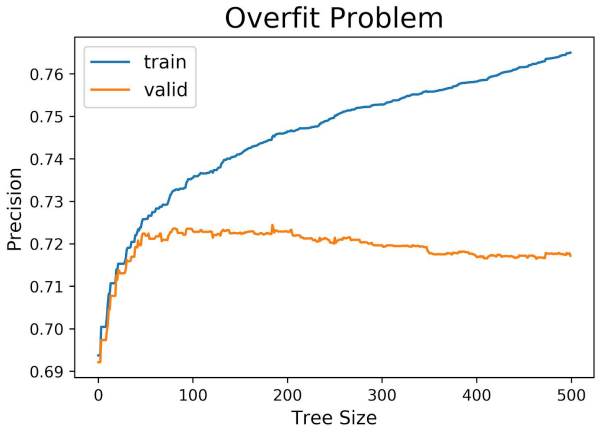
- **Using 5-Fold cross-validation to tune the parameter of decision tree**

Because the cross validation of 10 fold took a lot of time, we used the cross validation of 5-fold to adjust the parameters and find the parameters that could make the cross validation result the best. We made min_impurity_decrease=0.0007 and max_leaf_nodes=80 and min_sample_leaf = 5.

- **Using PCA(Principal components analysis) to optimize features**

I used principal component analysis to construct the data of six components, and I found that it greatly improved the operation speed and the results were better than before.

6. What performance did the first version of your classifiers achieve on the validation dataset (in cross-validation) and on the test dataset? Please comment on the performance of the classifier. (15 points: 5 points for performance, and 10 points for comments).



The method of checking overfitting was to compare train scores and validation scores with different parameter. From the above figure, we could find that when the tree size was more than 60, the train score was increasing but the validation score was decreasing, which was overfitting. Besides, it was easy to find that the decision tree classifier without parameter was a special example of overfitting(train score = 1, validation score = 0.6506). We could prevent overfitting by tuning the parameters to observe the score of cross validation.

7. What performance did you achieve on the validation dataset (in cross-validation) and on the test dataset after your modifications? Please, try to explain the gains. (15 points: 5 points for performance, and 10 points for explanation)

Performance:

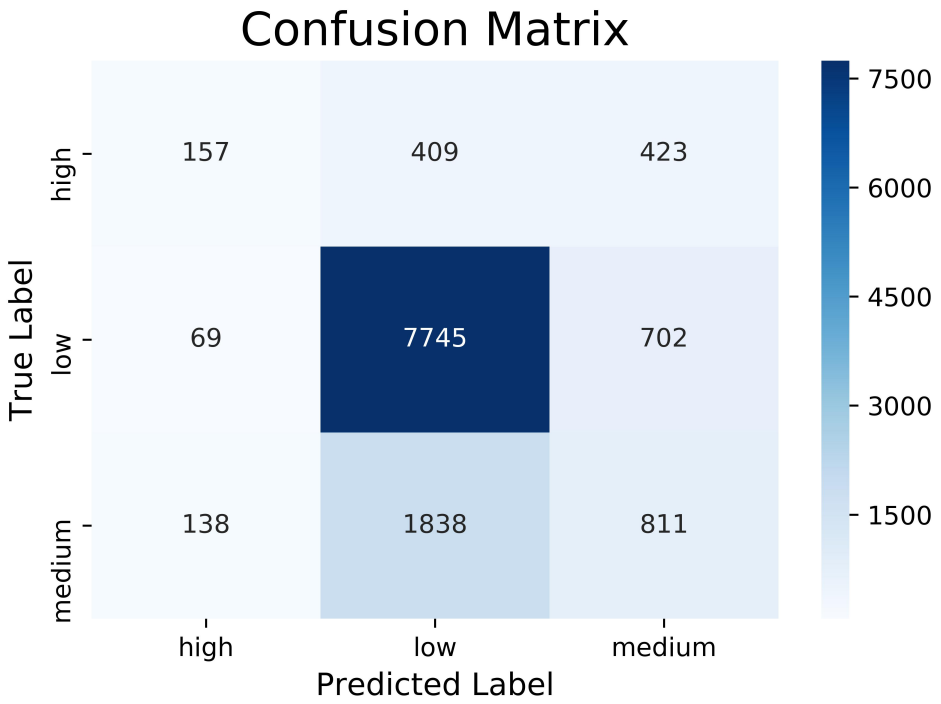
	Score of Validation	**Score of Kaggle**
Tuning parameters	0.7125	0.8042
Adding new features	0.7148	0.7675
Using PCA	0.7053	0.6910

Explanation:

Firstly, Because we tuned the better parameters and added new and more relevant features, making the decision tree more fit the data, so it would have better effect. For example, after adjusting min_importance_increase, it will be more efficient each time it is split. **Secondly**, The new text feature using TFIDF was more accurate than the manual selected one-hot data. Besides, the new feature distance to the nearest subway was helpful. **Thirdly**, the method of PCA helped us to select optimize feature.

8. Evaluate one additional evaluation metrics mentioned in class on the validation dataset. Which metric did you use? What were the results? How do these results compare to the results for multi-class logarithmic loss? (10 points)

- **Confusion matrix:**



I used the confusion matrix, which tells us whether the prediction result of each target. From this figure, we can observe the predicted results more intuitively than multi-class logarithmic loss. For example, we could find that some low levels are predicted as medium levels, therefore, we could find more features which could be used to distinguish these two levels. However, the confusion matrix could not directly tell us whether the classifier was improved or not. We check the improvement of the classifier by the result of multi-class logarithmic loss.

9. Bonus (10 points): You can combine your data with other, related datasets to create additional relevant features, for example, based on the nearby subway stations and malls. Which additional features did you create? By how much did these features improve the performance? If you do not train two different versions of your classifier (with and without the additional features), what evidence do you have that the additional features helped?

We found the dataset of New York's subway station and created the distance for the house to the nearest subway station. The distance feature did not improve the validation score but improve the score on the kaggle.(from 0.7856 to 0.7675) If we do not train two different versions of your classifier, we could find the correlation coefficient or f-score to evaluate the additional features. If the features had high score, we thought they would be helpful.

Part 2: Logistic Regression(Yizhou Chen)

1. Which features did you select for your classifiers? Please comment on the reason for your feature selection. If you choose to work on the bonus question, you can add your features extracted from external datasets at this step. (5 points)

- Selected features: *bathrooms, bedrooms, latitude, longitude, price*
- Derived features: *manager_id_num, photo_num*
  - Turned the string *manager_id* into a numerical variable *manager_id_num*
  - Added the number of photos *photo_num*
- Extracted features:
  - Tfidf vectors of features and description
  - doc2vec vectors of description
  - some words selected manually
- Additional features:
  - The distance to the nearest subway station

Comment:

I selected the features that makes sense to a house quality. People may choose a house based on the location, price, facilities, and the manager who sells this house. I abandoned building _id since as an incomprehensible string it doesn't make much sense to people. The display address is too abstract for us to use in classification, and the time the post is created is not so significant as other attributes. People will also try to know a house by description and photos, so I selected I vectors extracted from text and the number of photos. Considering that public transportation may be important when people choosing a house, we added the distance to the nearest subway station.

2. What Python or R libraries did you use for your classifiers? (5 points)

Python scikit-learn library

3. How did you perform cross-validation? Please describe the procedure. (10 points)

I used the KFold method: **from** sklearn.model_selection **import** KFold.

I set the parameter *n_split=5* to split the original training data into 5 same-size sets. In each of the 5 iterations, pick a different 4 of the 5 sets as trainning data and the rest 1 set as validation data. The 5 iterations produce 5 validation scores, and then average the results to get a evaluation for the current model.

4. What performance did the first version of your classifiers achieve on the validation dataset (in cross-validation) and on the test dataset? Please comment on the performance of the classifier. (15 points: 5 points for performance, and 10 points for comments).

Since the numerical attributes, Tfidf vectors, and doc2vec vectors are different kind of features, so initially I trained three separate logistic regression classifier for these three kinds of attributes.

*LogisticRegression(penalty='l2', C=1.0, class_weight='balanced',solver='liblinear', multi_class='ovr')**

- **Selected & Derived Attributes:**
  - Cross-validation(5-fold) score: 0.6499
  - Kaggle evaluation score for test data: 0.80859
- **Tfidf Vector of Features :**
  - Cross-validation(5-fold) score: 0.6527
  - Kaggle evaluation score for test data: 0.83040
- **Doc2Vec Vector of Descriptions:**
  - Cross-validation(5-fold) score: 0.6623
  - Kaggle evaluation score for test data: 0.85372

Comment:

The results are acceptable. The vectors works better than simply the numerical attributes on the validation data, however, the simple numerical attributes achieved the lowest multi-class logarithmic loss. But in fact, none of them reached a satisfactory accuracy separately. I used the one-vs-rest method to deal with multi-class classification problem, it is easy to interpret, but may not be so accurate. I choose the default solver for gradient ascent, there may exists better solvers for these training data. Besides, the penalty for regularization and the regularization term may be tuned to improve this performance, in case there exists over-fitting.

5. What actions did you take in order to improve your classifiers? You can modify your dataset or the parameters of your classifier. Please record your modifications in your report. (30 points: 10 points for each improvement)

- **Tune the parameters**

I tried different way to deal with multi-class and different solvers to deal with the gradient iteration. I also tuned the regularization term. Besides, I tried to remove the balanced class-weight and increase the maximal number of iterations:

- Remove *class_weight='balanced'
- Change the method to deal with multi-class classification
- multi_class='multinomial'
- Change the solver of the gradient ascent iteration solver='sag'
- Tune the parameter C=10.0
- Increase the maximum iterations max_iter=500

LogisticRegression(penalty='l2', C=10.0, solver='sag', multi_class='multinomial', max_iter=500)

- **Modify the dataset**

  For the Tfidf and Doc2Vec vectors:
  - change the length of the vectors
  - slightly adjust several other parameters

  I tried to several methods to combine the three features or the results of the three classifiers to get improvement:
  - Add a few feature words extracted manually from features and descriptions to the original attributes. If the word exists in this record, then set the corresponding attribute value to be 1, otherwise 0.
  - Set the probabilities obtained from the three original classifiers as new attributes (9 attributes in total), and do logisitic regression classification on the new 9 attributes.
  - Turn the Tfidf vectors of features and descriptions into attributes (number of attributes equals to the length of the vector) and let the Tfidf value be the attribute value, combine them with the original attributes.

6. What performance did the first version of your classifiers achieve on the validation dataset (in cross-validation) and on the test dataset? Please comment on the performance of the classifier. (15 points: 5 points for performance, and 10 points for comments).

I checked over-fitting by comparing the score of the training set and the validation set while doing 5-fold cross validation. If the score of training set increases but the score of validation set decreases, then over-fitting occurs.

I tried *penalty=none* which means no regularization, and I tried to tune other parameters. But it seems that over-fitting didn't occur. The iteration of gradient always converges and the scores of training set never got dramatically high and the score of validation only fluctuated slightly.

To avoid over-fitting, I set the parameter *penalty=l2* to apply a regularization, and tuned the parameter *C* to adjust the regularization strength, until I get the best score on the validation set as I can.

7. What performance did you achieve on the validation dataset (in cross-validation) and on the test dataset after your modifications? Please, try to explain the gains. (15 points: 5 points for performance, and 10 points for explanation)

Performance:

- After doing logistic regression on the probabilities:
  - Cross-validation(5-fold) score: 0.7008
  - Kaggle evaluation score for test data: 0.72949

- After combine vectors and attributes:
  - Cross-validation(5-fold) score: 0.6971
  - Kaggle evaluation score for test data: 0.69609

Explanation:

For the parameters of the classifier, choosing multinomial instead of one-vs-rest to deal with multi-class classification provides a higher accuracy, and the solver 'sag' is good at dealing with large amount of data. Since I over-fitting didn't occur, I increase***C** to reduce the strength the regularization, meanwhile increase the maximum iteration for the gradient ascent process. Since the number of records labeled low is almost ten times the number of records labeled high in training data, I set the class-weight to be balanced at first, however, this leads to a low accuracy and high logarithmic loss. I finally removed the class weigh. After the parameters are tuned, I gained a better score.

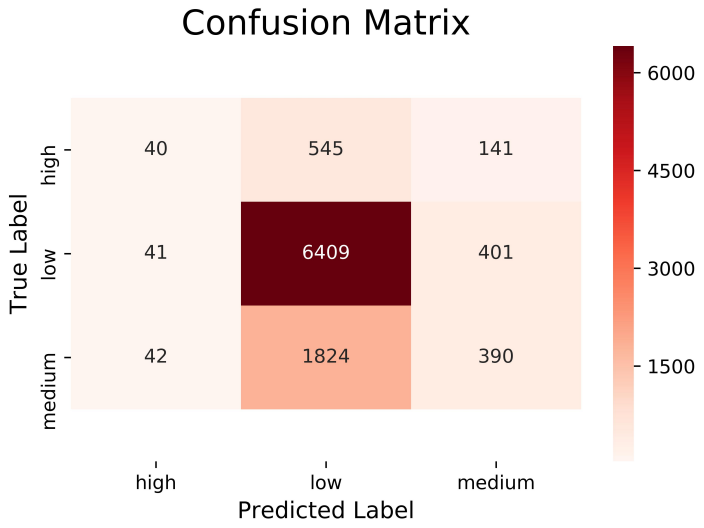
I got the best validation score by doing a second logistic regression on the results of the first three logistic regressions, which is similar to what ensemble classifiers do. A second classification combined the results of the different features, made the three separate classifiers complement each other, which leads to a better result.

I got the lowest logarithmic loss by just concatenate the Tfidf vectors with the numerical attributes. I tuned the size of the vector, and finally I choose 80 for Tfidf vector of features and 80 for Tfidf vector of features. So I had 169 features at last. Concatenation provide the most complete information, which leads to the greatest decrease in the logarithmic loss, although the original 9 attributes may loss their significance in the 169 features. Probably it's the text that provides the highest discrimination in the classification.

8. Evaluate one additional evaluation metrics mentioned in class on the validation dataset. Which metric did you use? What were the results? How do these results compare to the results for multi-class logarithmic loss? (10 points)

I calculation the recall, precision and the confusion matrix.

- results of recall and precision in 5-fold cross-validation:
  - macro recall 0.3738, 0.3776, 0.3777, 0.3792, 0.3879
  - macro precision 0.5132, 0.5481, 0.5379, 0.5212, 0.4917
- A confusion matrix:



The multi-class logarithmic loss is a complicate and comprehensive evaluation method for the performance of a classifier. We can directly judge whether the performance is improved by checking if the loss is reduced. The recall, precision and confusion matrix are more interpretable and intuitive. The confusion matrix provide more information about each class, we can see for which classes we made better predictions and for which classes we were not doing well.

9. Bonus (10 points): You can combine your data with other, related datasets to create additional relevant features, for example, based on the nearby subway stations and malls. Which additional features did you create? By how much did these features improve the performance? If you do not train two different versions of your classifier (with and without the additional features), what evidence do you have that the additional features helped?

additional feature: distance to the nearest subways station.

- performance improved:
  - When I add it to the attributes set without tfidf and doc2vec vectors
    Cross-validation(5-fold) score: from 0.6918 to 0.6922
    Kaggle evaluation score: from 0.72784 to 0.72691
  - When I add it to my final classifier which has total 169 attributes among which 160 attributes are Tfidf values, the validation score only improved slightly and the Kaggle evaluation score didn't change.

So the conclusion is that, this feature indeed help, but one attribute's contribution seems too small compared to 160 Tfidf values.

Part 3: Support Vector Machine(Luoxi Meng)

1. Which features did you select for your classifiers? Please comment on the reason for your feature selection. If you choose to work on the bonus question, you can add your features extracted from external datasets at this step. (5 points)

I use both the result of original features from the dataset and the tfidf result from text feature extraction.

- **Original Data:** 'bathrooms', 'bedrooms', 'created', 'latitude', 'longitude', 'price'
- **Transformed Data:** 'manager_id_num', 'building_id_num', 'photo_num', 'price_per_bedroom', 'price_per_bathroom',
- **Text Data:** tfidf of "description" and "features"

Comments:

The **original features** include "number of bedrooms", "number of bathroom", "price", "latitude", "longitude", etc. And I dropped some features like "listing_id", "rec_id" which is key of this listing or record.

As for text feature (tfidf), I decrease the parameter 'max_featrue" to 20, and add the 40 (20 from "description", and 20 from "features") as additional features to the original feature set.

There should be a balance between using the text features to raise the accuracy and avoiding putting on too much weight on text features (when the vector is too long) that decrease the weight of some important features like "price". So, 20 is a good choice if we can carefully weight the features later.

2. What Python or R libraries did you use for your classifiers? (5 points)

Python Library: sklearn

3. How did you perform cross-validation? Please describe the procedure. (10 points)

I use 5-fold cross-validation in a loop with KFold() method. KFold splits the index of train data into two parts(1/10 for validation dataset, 9/10 for training dataset). Then, we used the training dataset to learn a model and the validation dataset to get the score as an evaluation of the model.

Then I evaluate the accuracy with the formula: *Accuracy = mean_score (+/- std_score * 2). (std stands for standard deviation)*

Then when tuning the parameters, I use GridSearchCV() to perform a 5-fold validation for all the combinations of the parameter grid.

4. What performance did the first version of your classifiers achieve on the validation dataset (in cross-validation) and on the test dataset? Please comment on the performance of the classifier. (15 points: 5 points for performance, and 10 points for comments).

The first version of my classifier achieves a 5-fold cross-validation score of 0.6937, and a logloss of 0.87.

Comments: This model is learnt based on the default parameter of SVC, which is (kernel='rbf', C=1, gamma='scale'). Meanwhile, I have not yet done much on the data and only use the tfidf data which is a sparse matrix. But actually it comes out not bad, which indicates that SVM is a classifier of high accuracy and not so bias-prone like decision tree.

5. What actions did you take in order to improve your classifiers? You can modify your dataset or the parameters of your classifier. Please record your modifications in your report. (30 points: 10 points for each improvement)

- **Decrease the number features from tfidf from 300 to 40, and drop those highly correlated and those insignificant to the final result.**

I evaluate the correlation of each 2 features with items in X.corr() and drop the features that are highly correlated to another one. Additionally, with the library statsmodels.api I evaluate how a feature helps to the regression of the result of y_train and the drop those insignificant features (though this idea was not implemented finally, resulting from the observation that it might lead to overfitting.

- **Apply PCA to reduce the data into main components.**

PCA helps to reduce the noise of data, and make the implementation more robust. I use 3 components in this case. And this greatly improve the efficiency in the model training process.

- **Using PCA(Principal components analysis) to optimize features**

GridSearchCV is a method from sklearn.model_selection that provides an approach to do the exhaustive search with all the combinations of the parameter in a "parameter grid". I use this method and set the "scoring" parameter of it to "accuracy". Then all the parameter combinations in the grid are evaluated by their accuracy with 5-fold cross validation. Then we can find the best parameters.

6. How did you check whether any overfitting occurred during your training?Did you observe overfitting? What did you do to avoid overfitting? (10 points)

- **Observe overfitting:**

In my implementation, I observe when I use (kernel='rbf', C=1, gamma=1e-3), I achieve a high score (0.997) on the training set, but the logloss score on Kaggle was disappointing (only 0.85, a little bit lower compared without other combination of parameters). The opposite case is that when I am using the parameters

(kernel='sigmoid', gamma='auto', C=1), the score on the training set is only 0.69374. But this time the logloss score evaluated in Kaggle is 0.79075. So, in this 2 cases, the one with higher accuracy on training set failed to be the one with higher accuracy on the test set, which is the case of overfitting.

- **Avoid overfitting**

In SVC, the parameter C trades off between the accuracy against maximization of margin. Thus, by setting an appropriate C (in my case, C=1) is a way to avoid overfitting. And

7. What performance did you achieve on the validation dataset (in cross-validation) and on the test dataset after your modifications? Please, try to explain the gains. (15 points: 5 points for performance, and 10 points for explanation)

Performance:

	Score of Validation	**Score of Kaggle**
First attempt	0.6937	0.8767
Change kernel to 'sigmoid'	0.6634	0.8078
Tuning parameter 'C	0.7390	0.7905
Modifying Dataset (apply PCA and StandardScaler)	0.7930	0.7860

Explanation:

My first attempt is to change the kernel from default 'rbf' to 'sigmoid'. This gives constant result of score. We notice that this kernel has a better performance on the test dataset, but worst performance in the validation, which may be a clue for overfitting.

Then I try to tune the parameter C with GridSearchCV, by parsing C = [0.001, 0.01, 0.1, 1] to the parameter grid. As C behaves as a regulation parameter in SVM and lower C will encourage a larger margin, by trying to decrease C, we all a penalty for the misclassification, and thus, slightly fixing the overfitting problem. We then see this gives a raise in both cross-validation and test set.

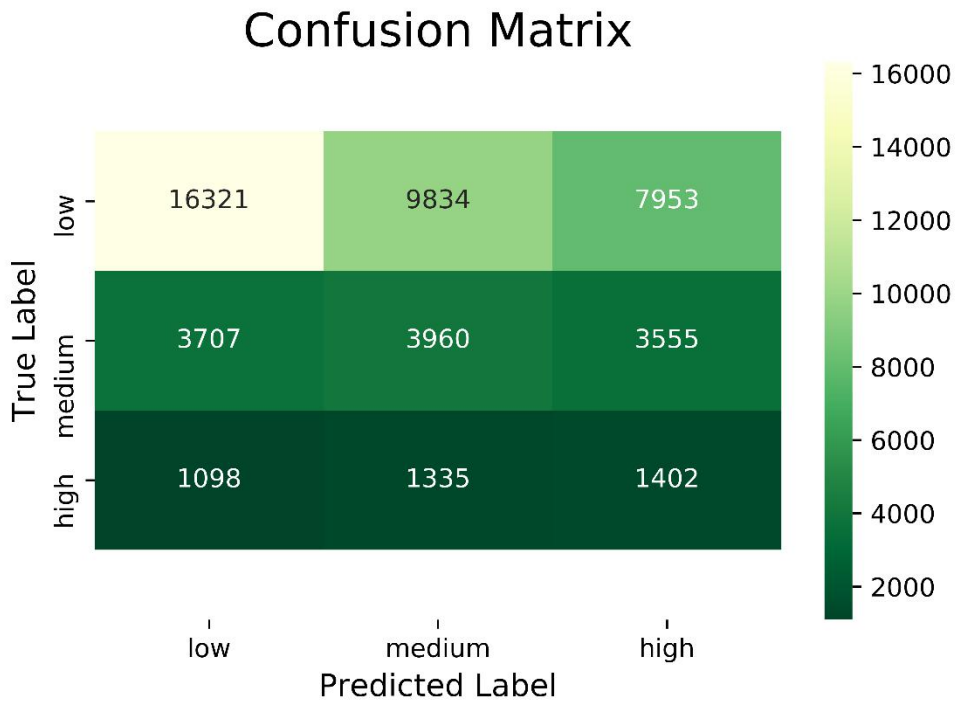
Then I try to modify the dataset, make the features better predict the class label. The main method is dropping some correlated columns and insignificant features (as already detailed above), and apply PCA and StandardScaler. This works and improve the result a little bit for both cross validation and test data.

Comments:

In general, there is no great but slightly improvements on the result if tuning the hyperparameters (C, gamma) step by step. And since the training of SVC is quite time-consuming, I did not find a quite efficient way to decrease the logloss and better predict the test data.

8. Evaluate one additional evaluation metrics mentioned in class on the validation dataset. Which metric did you use? What were the results? How do these results compare to the results for multi-class logarithmic loss? (10 points)

- **Confusion matrix:**



I use confusion matrix to evaluate the precision and recall of the performance on validation set. From the confusion matrix, we notice that the performance of prediction on the "low" class is much better than "medium" and "high". The performance on the "high" class is the worst.

Github Link:
https://github.com/Wukkkinz-0725/CMPT459_Project_RentalListingInquiries/tree/master/Milestone2