CMPT459 Milestone 3

Part 1: Random Forest(Jinze Wu)

1. What classifiers did you use for milestone 3? (5 points)

Random Forest Classifier

2. 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', 'distance_subway'
- **Text Data:** TFIDF Results of 'Features' and 'Descriptions'(One-hot Text Data Before)

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 *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.

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).

Random Forest Classifier(No Parameter)	**Random Forest Classifier(max_depth = 20)**
Validation: The average score(10 Fold) = 2.008	Validation: The average score(10 Fold) = 0.6868
Test: The score on the kaggle = 1.358	Test: The score on the kaggle = 0.7611

Comment:

In order to get better score on the kaggle, we choose log_loss() as the criteria to evaluate the classifier. Therefore, we use function log_loss() in cross-validation. We found that the result of the classifier without parameter looked not good on either validation data or test data. After adding simple parameters, the random forest classifier's processing results for validation and test were very different: the results on validation looked good, but the results on test looked plain. I think the reason of the different results between two datasets might be the incompletely processed dataset.

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. (20 points)

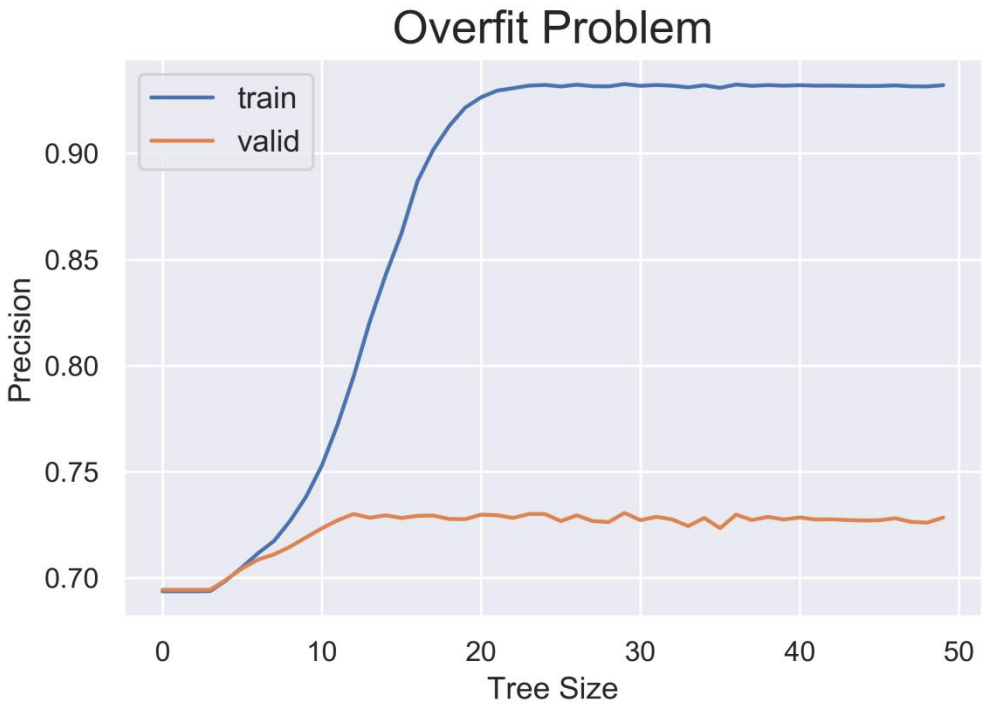
- **Adding new feature: distance to the nearest subway station, 20 most important TFIDF of features and descriptions(10 features, 10 descriptions)**

  I think the numerical text data will be very helpful, so I performed TFIDF processing on the text data and added the filtering part to the data set.
- **Using PCA(Principal components analysis) to optimize features**

  I used principal component analysis to construct the data of 30 components, and I found that it greatly improved the results.
- **Tune the parameter of RandomForestClassifier**

  We used log_loss as the criteria to evaluate the result of prediction. Firstly, we identified an n_estimators that would stabilize the results, then we used a method similar to GridSearchCV to find the optimal parameters (two layers of ‘For Loops). We found that when the max_depth = 21 and min_samples_split = 40, the tree had the best performance.

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)



The method of checking overfitting was to compare train scores and validation scores with different parameter(max_depth). As we all knew, the based estimators of random forest classifier was decision tree classifier. If the decision tree was not deep enough or too deep, the classifier result would be fit bad or overfitting.

From the above figure, we could find that when the depth of the trees was more than 12, the train score was increasing but the validation score was stable, which meant that the random forest classifier would not overfitting because it had random property. Therefore, we could find that when the based estimator was overfitting, the random forest classifier would prevent it from overfitting. But we would better look for the best parameters to make the tree fit best, rather than over fitting parameters.

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**
Improved Result	0.6247	0.6228

Explanation:

Firstly, we add new TFIDF text feature which was more accurate than the manual selected one-hot data and new numerical data which was the distance to the nearest subway station.

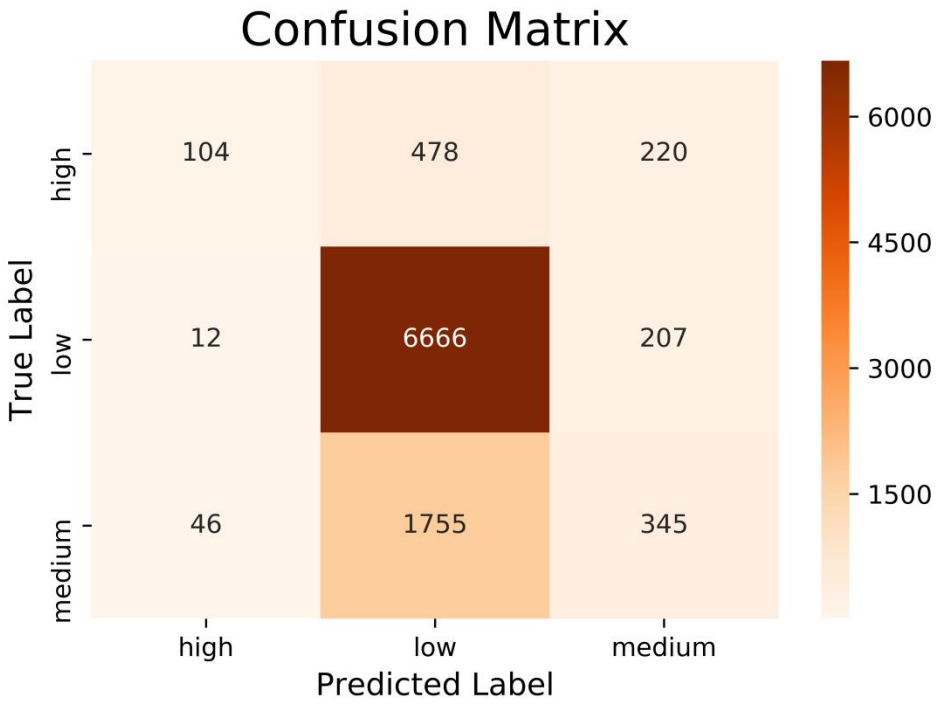
Secondly, because the lots of values of TFIDF data will be zero which would greatly influence the results, so we use PCA to construct 30 new attributes.

Thirdly, we tuned the better parameters to make the based estimators more fit the data, so it would have better effect.

We found that the score of validation and kaggle were very close after improving. And, after three steps of improvements, we improve the score **from 0.8398 to 0.6228.**

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. Compare your new classifier with the classifiers used in milestone 2. Try to explain the difference between the performance of the classifiers and the gains of the milestone 3 classifiers. (10 points)

- **The scores on the kaggle:**

	First Version	Improved Version
Decision Tree	**0.8449**	**0.6910**
Random Forest	**0.7611**	**0.6228**

The difference between the performance of the classifiers:

According to the table, the performance of random forest is better than that of decision tree when classifying the dataset. And random forest performance is very stable.

The gains of Random Forest Classifier:

1. There is no overfitting problem in random forest classifier.
2. RandomForestClassifer provides a more reliable feature importance estimate.
3. The decision tree classifier has a high variance, but low bias. But because we considers all the trees in random forest so that we have a low bias and moderate variance model, which will be better than decision tree.
4. Because of the above reasons, the random forest classifier can make me get the accurate classification results more effectively.

10. Bonus (10 points): The top three groups that achieve the best performance on the test dataset (computed by Kaggle) will get 10 bonus points.

My best score: 0.6228

Part 2: XGBoosting (Yizhou Chen)

1. What classifiers did you use for milestone 3? (5 points)

XGBoosting

2. 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:

- Selected features: *bathrooms, bedrooms, latitude, longitude, price, created*
- 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*
  - Calculated the price per bedroom *price_per_bdr*
- Extracted features:
  - Tfidf vectors of features and description

Then, Use extremely randomieze trees classifier to do feature selection, and reserved the top 100 important features.

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.

Having over 170 features, there may be features have mutual information and some may not be useless for classification, so I used *Extra-Trees* method to select the most important 100 features.

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

- K-fold 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 training 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.

- xgb.cv() method: **import** xgboost **as** xgb

Set the parameter *cv=5*, and the 5-fold validation will be performed in each iteration until it find the best number of estimators.

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).

Initial version:

initial_xgb = XGBClassifier(booster='gbtree', objective= 'multi:softmax', num_class=3, learning_rate =0.1, n_estimators=100, max_depth=12, min_child_weight=1, gamma=0, seed=27)

Performance:

	Accuracy of 5-fold Validation	**M_log_loss Score of Kaggle**
Initial version	0.73417	0.60360

Comments:

The results are good, compared to other classifiers. I choose the default value for many parameters including *subsample, gamma, alpha* and so on. But set a better value for these parameters may help to restrict the trees, obtain better split choice and get better performance. And 100 estimators may not be enough, increase the number of estimators may further improve. And we also don't know the *max_depth is suitable*, it may be too small and lead to underfitting. Besides, since the average accuracy on the training set achieved 0.96017, there's high probability of 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. (20 points)

Use *GridSearchCV* to tune the parameters:

- Param_test1 = { '*max_depth*': [3,5,7,9], '*min_child_weight*':[1,3,5] } - got (9, 5)
- Param_test2 = {'*max_depth*': [8,9,10], '*min_child_weight*':[4,5,6] } - got (9, 6)
- Param_test3 = {'*gamma*': [i / 10.0 for i in range(0,5)] } - got gamma=0.1
- Param_test4 = {'*subsample*':[i / 10.0 for i in range(6,10)], '*colsample_bytree*': [i / 10.0 for i in range(6,10)] } - got (0.9, 0.7)

Use *xgb.cv* to find the best *n_estimater*:

- Reduce *learning_rate* to 0.05
- Increase *n_estimators* to 800

Modify Dataset:

- Use Extra-Trees to select the most important features
- Add additional Features *distance_to_sub*

Model fusion

- Use the trained XGBoosting to extract features, and then do Logistic Regresion
- Let XGBoosting and RandomForeset be base classifiers and Logistic Regression be meta classifier and apply stacking

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)

I checked overfitting by comparing the *mlogloss* (multi-class logarithmic loss) of the training set and the validation set while doing 5-fold cross validation. If the loss of training set decreases but the loss of validation set increases, then overfitting occurs.

	train-mlogloss-mean	train-mlogloss-std	test-mlogloss-mean	\
0	1.091271	0.000031	1.092196	
1	1.083734	0.000138	1.085700	
2	1.076574	0.000261	1.079569	
3	1.069450	0.000397	1.073374	
4	1.062591	0.000371	1.067460	
5	1.055794	0.000429	1.061596	
6	1.049393	0.000617	1.055993	
7	1.042802	0.000828	1.050346	
8	1.036299	0.000976	1.044738	
9	1.029901	0.001064	1.039215	
10	1.023300	0.001084	1.033626	
491	0.338588	0.001242	0.592535	
492	0.338241	0.001178	0.592491	
493	0.337865	0.001165	0.592446	
494	0.337478	0.001169	0.592405	
495	0.337120	0.001154	0.592361	
496	0.336767	0.001164	0.592321	
497	0.336438	0.001184	0.592282	
498	0.336097	0.001253	0.592252	
499	0.335764	0.001239	0.592206	

I observed overfitting in my initial classifier. My initial classifier got the 0.96 accuracy on the training set but only 0.73 accuracy on the validation set. Compared to my improved classifier, whose accuracy on training set is 0.90 and accuracy on validation set is 0.75, the initial version has higher accuracy on training set but lower accuracy on validation set, which showed overfitting.

To avoid overfitting, I tuned the parameters *min_child_weight* and *max_depth* to limit the minimum weighted sum of the children and the maximum depth of the trees, and also tuned the parameter *gamma* and *alpha* to adjust the strength of regularization, until I got 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)

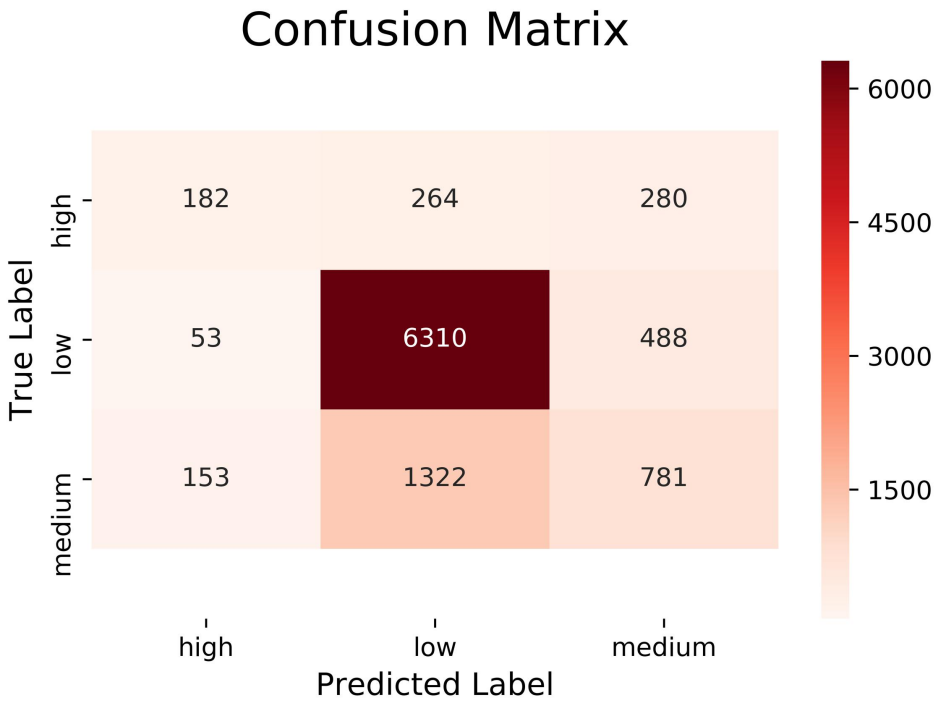
	Accuracy of 5-fold Validation	**M_log_loss Score of Kaggle**
Improved version	0.74860	0.57766

Explanation:

- Better choice for *max_depth* and *min_child_weight*: These two parameters have the most strong influence on the classifier. Limiting the maximum depth and minimum child weight may avoid underfitting and overfitting.
- Better choice for *gamma: gamma* is closely related to loss function. Restricting the lower-bound of the loss required for splitting may bring improvement, and can also help avoid overfitting.
- Better choice for *subsample* and *colsample_bytree*: Increased *subsample* avoided underfitting and make the classifier fits the data better. Decreased *colsample_bytree avoided overfitting.* The two parameters together help to reduce variance.
- Selecting the most important features reduces the dimension of the feature space, improve the training efficiency and also improve the performance.
- Additional features: derived features *price_per_bdr*, *price_per_btr*, and additional feature *distance_to_subway* provided extra important information
- The fusion didn't help much.

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:**



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. Compare your new classifier with the classifiers used in milestone 2. Try to explain the difference between the performance of the classifiers and the gains of the milestone 3 classifiers. (10 points)

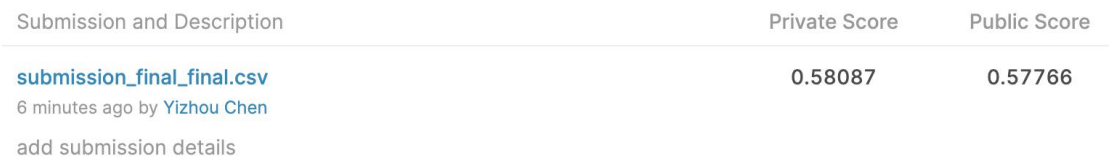
The scores on the kaggle:

	First Version	Improved Version
Logistic Regression	**0.80859**	**0.69609**
XGBoosting	**0.60360**	**0.57766**

- Logistic Regression is linear while XGBoosting is non-linear. Logistic Regression is based on the assumption that the data is linearly separable and the dicision surface is a hyperplane. XGBoosting can split infinitely and generate a more complex decision surface, which is a combination of sub-surfaces parallel to the feature axes. So XGBoosting may reduce bias and have better performance.
- Logistic Regression treats every featual equally, while XGBoosting always seeks the best split point with lowest loss which means it has the ability to seek a good combination of those more important features.
- The regularization method of Logistic Regression and XGBoosting is different, so they have different differentiability and optimizer, which leads to different performance.
- Logistic Regression only have 1 estimator, while XGBoosting have many estimators. The weight of misclassified traning examples will be increased in a new classifier that corrects for the bias on these particular instances.
- XGBoosting, as an ensemble classifier, also uses *subsample* and *colsample_bytree* to reduce variance.

10. Bonus (10 points): The top three groups that achieve the best performance on the test dataset (computed by Kaggle) will get 10 bonus points.

My best score: 0.57766



Part 3: Gradient Boosted Decision Tree(Luoxi Meng)

1. What classifiers did you use for milestone 3? (5 points)

Gradient Boosting Decision Trees (GBDT)

2. 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)

- Original features: I reserve 'bathrooms', 'bedrooms', 'created', 'latitude', 'listing_id', 'longitude', 'price'.
- Transformed data: I add 'diff_rooms' (the difference of the number of bedrooms and bathrooms of each listing), 'manager_id_num' (integer value to identify the same manager), 'photo_num' (number of photos provided in each listing), 'price_per_bedroom', 'price_per_bathroom'.
- Text data: I extract 50 features from tfidf of 'description', and 50 features from tiidf of '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_featrues" to 50, and add the 100 (50 from "description", and 50 from "features") as additional features to the original feature set. And I drop some columns that is highly correlated with another one.

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".

The resulting number of features used for classification is basically 100 after correlated column dropping.

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)*

When tuning the parameters, I use StratifiedShuffleSplit() to split the training data into training set and validation set and use GridSearchCV() to perform a 5-fold validation for all the combinations of the parameter grid.

All the cross validation takes 'neg_log_loss' as scoring policy.

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).

- Performance:
  CV score: 0.690 (+/- 0.005);
  Kaggle score: 0.83833
- Comments:
  For my first attempt of GBDT, I use the tree-specific parameters that my teammate used in Milestone2 (decision tree) and set learning_rate=1, n_estimators=10. In Milestone2, this set of parameters achieves a high score in Kaggle (0.69..). However, it is quite interesting that it fails to give a high score in this case.
  I think the reason could be:
  - learning_rate=1 is too fast, and cause a pre-converge.
  - When GBDT boosting too fast, the boosting process will greatly decrease the variance but increase the bias.

  So, obviously, the set of parameters that works in decision tree will not work in GBDT. However, this does give a good intuition about parameter tuning.

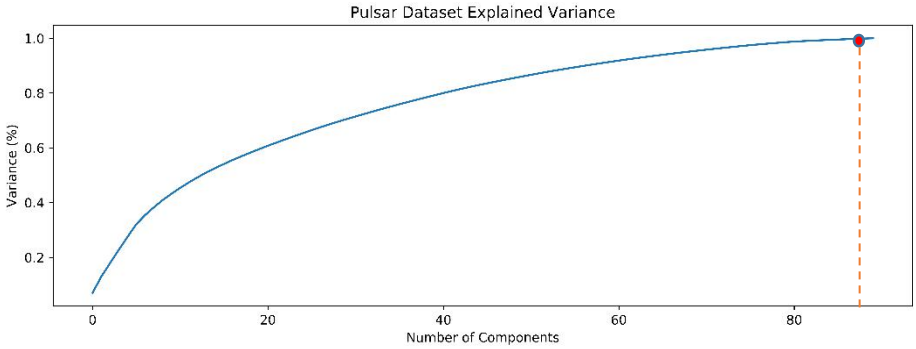
5. What actions did you take in order to improve your classifiers? You can modify your dataset or the parameters of your classifier. Please record your modifications in your report. (20 points)

- Dataset Modification
  - Drop high correlated columns

  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. To set the number of principle components, there is a trade-off between the robustness of small number of components and the reservation of variance of original features. I plot a figure to find an optimal value.



Accoring to the figure, I choose n_components=90.
Compared with the classification without PCA, we can say for sure that PCA transformation helps to improve buth the efficiency and accuracy of classification

- Parameter Tuning
  - Tool:
    The parameter tuning process is based on the functionality of GridSearchCV(), which applies an exaustive search on the parameters provided in a parameter grid.
  - Order of Parameter Tuning:
    We tuning the parameter in the descending order of its impact on the model, which means:
    1) Tune n_estimators, learning_rate
    2) Tune max_depth, min_samples_split
    3) Tune min_samples_leaf
    4) Tune max_features

    We fix the value with a higher impact on outcome first, and then fix its value to tune other parameters.
    Finally, we increase the n_estimator and decrease learning_rate proportionally to improve robustness.

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)

Yes.

- Observation

I observe overfitting in the case when cross validation on the training give a quite good result when I set n_estimators=100, learning_rate=0.2 (logloss=0.591834), However, this model fails to give a good result in Kaggle when applied to the test set (score=0.68491). In another case, use learning_rate=0.1 to replace 0.2 give a better result in Kaggle (score=0.66507), but the score in cross validation is only 0.611727.

- Overfitting Avoidance

In fact, as a boosting method, GBDT is designed for decreasing the bias, not the variance. And it is quite robust. But we can still do something to avoid overfitting. And the most important thing to do is set the tree-specific parameters. For example,

max_depth, min_samples_split, min_samples_leaf. These parameters prevent the base estimators, i.e. decision trees, to grow too deep, which is exactly the cause of overfitting.
After carefully tuning the 3 parameters (max_depth, min_samples_split, min_samples_leaf), the training process becomes more efficient and robust. Meanwhile, this contributes to the final score as well.

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)

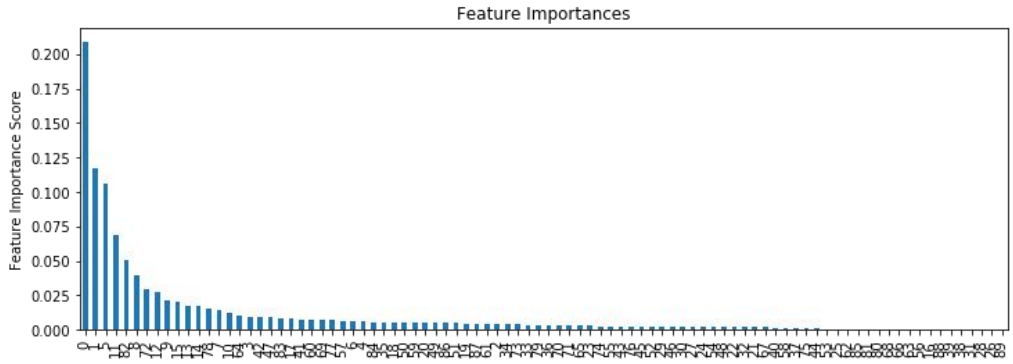
Performance:

	Score of Validation	**Score of Kaggle**
First attempt	0.690	0.838
Tuning boosting parameters	0.695	0.665
Tuning tree-specific parameters	0.703	0.652

Explanation:

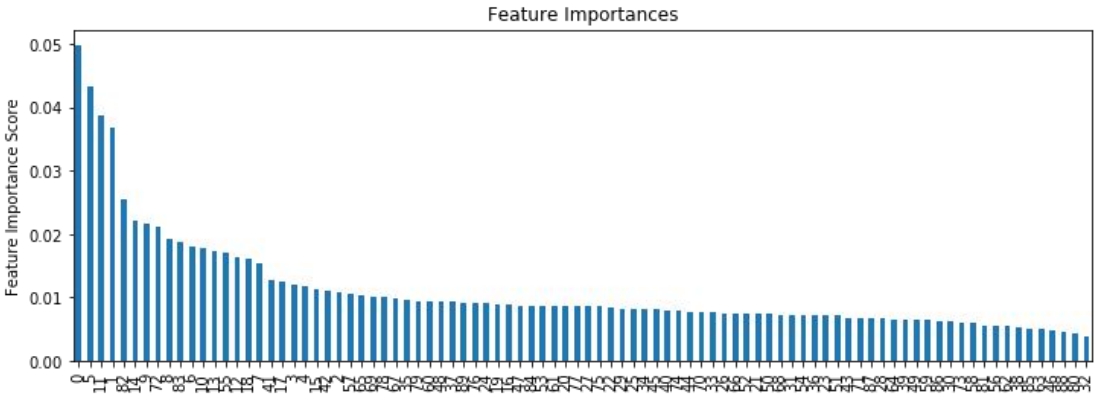
The order of parameter tuning is quite important. In my implementation, I tune the boosting parameters first and then tree-specific parameters.

My first attempt use the partial result of a best estimator for decision tree, with n_estimator=10, learing_rate=1.0. To present the result more explicitly, I plot a figure to show the weight of each features in the classifier. The feature importance in my first attempt is shown below. We can see that the highest weight reaches 0.2 (too high) and there are too many feature with weight of nearly zero. This is due to the high learning_rate, the learning process is terminated before the model converges.



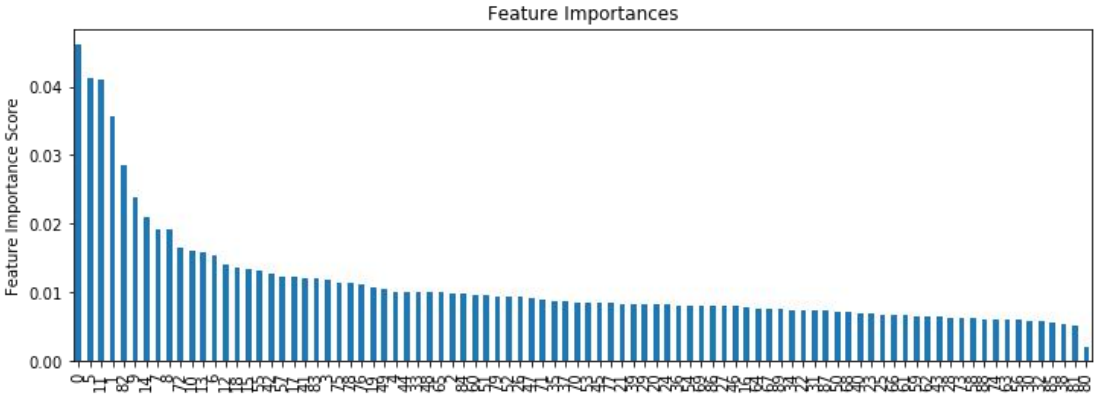
1st Attempt

Then I tuning the boosting parameters (n_estimators and learning_rate) with GridSearchCV and set 'n_estimators' in [40, 50, 60, 70] and 'learning_rate': [0.05, 0.1, 0.15, 0.2]. This helps to find the best pair of boosting parameters (in my case, n_estimators=70, learning_rate=0.1). And the plot looks much better (shown below)



After tuning boosting parameters

After that, I tuning the tree-specific parameters (max_depth, min_samples_split, etc.) This parameters define the behaviors of base estimators (decision trees) and help avoid overfitting.

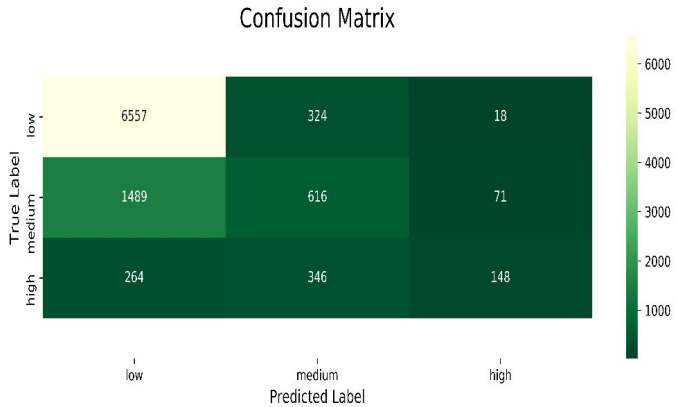


Comments:

From the parameter tuning process, we can see that the weight is much balanced after tuning, which indicates the slowing down learning rate and more consideration into other features instead of putting too much weight on a single one.

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



Comments:
Logloss takes into account the uncertainty of prediction based on how much it varies from the actual label. Confusion matrix presents the result more explicitly by providing the predict label and true label.
Obviously, confusion matrix does not take the probability of prediction into account.
But this 2 metrics can both distinguish good prediction from not so good ones. Good predictions minimize logloss, and value of good predictions will be mostly located in the diagonal.
For example, in my implementation, the prediction on 'low' is much accurate than 'medium' 'high' with a higher portion of data on the diagonal.

9. Compare your new classifier with the classifiers used in milestone 2. Try to explain the difference between the performance of the classifiers and the gains of the milestone 3 classifiers. (10 points)

Performance:

	First Attempt	Improved
SVC	**0.8767**	**0.7860**
GBDT	**0.8380**	**0.6523**

Explanation:
GBDT is an ensemble classifier that take decision tree as the base estimator. The goal of GBDT is to reduce the bias.
In Milestone2, I use SVM (SVC in sklearn) to learn the model. And I found the result of probability predictions from SVC is generally the same for each listing. It is high accurate (with mean Kaggle score 0.78) but it makes a strong assumption about the training data. So, it will tend to have bias and variance. Meanwhile, the training time of SVC is quite longer than GBDT (if n_estimators is not too large).

In Milestone3, GBDT gives a much better result, though the parameter tuning process is quite difficult compared with SVC. GBDT has a large set of parameters that have influence on each other. The boosting parameters n_estimators and learning_rate play important roles in giving a better result. By boosting the wrong predictors (setting higher weight), the leading predictor correct the wrong base estimators. That the main reason for the accuracy gain.

10. Bonus (10 points): The top three groups that achieve the best performance on the test dataset (computed by Kaggle) will get 10 bonus points.

0.6523

Github: https://github.com/Wukkkinz-0725/CMPT459_Project_RentalListingInquiries