Machine Learning Engineer Nanodegree

Capstone Project

Che-Chuan Li Udacity
January 18th, 2019

I. Definition

Project Overview

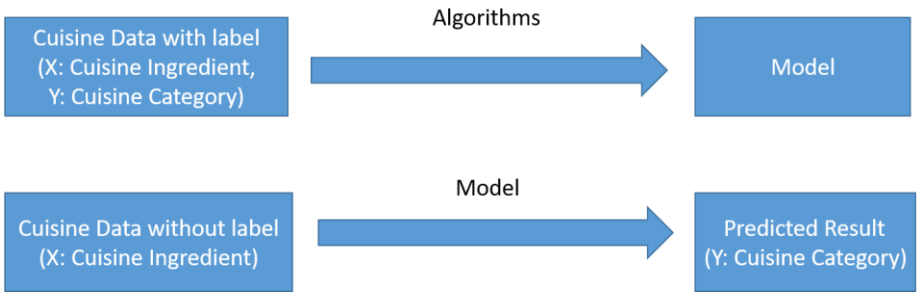
The goal of this project is “use recipe ingredients to categorize the cuisine”, which is from Kaggle competition: https://www.kaggle.com/c/whats-cooking-kernels-only. The dataset provided by Kaggle is from Yummly.

More specifically, given a list of ingredients of a cuisine, software program should automatically categorize the cuisine without human effort. The ingredients are like peppr and grape tomatoes, and the categories are like Greek and Indian. The dataset can be used by program is categorized cuisine dataset.

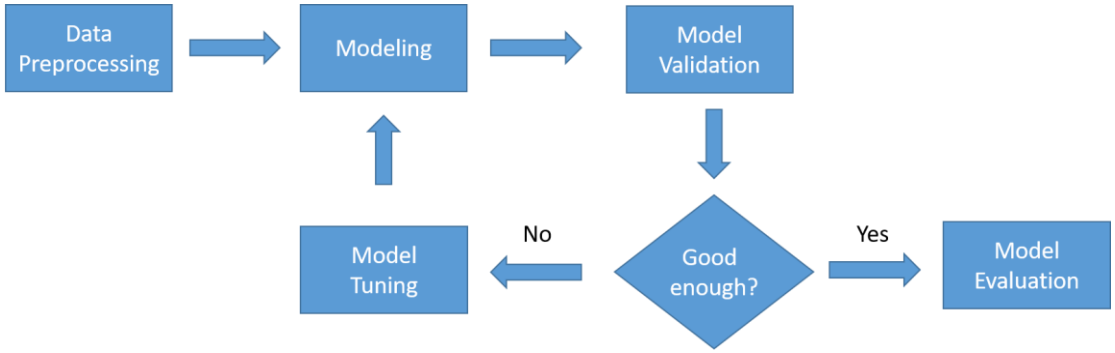
The program can be applied to product recommendation on ecommerce website or recipe recommendation on recipe sharing website. On ecommerce website, when customers put some ingredients into cart, ecommerce can recommend other products which they may interest based on the category which contains culture information, although the ingredients in dart in dart may be bought for one or multiple cuisines. On recipe sharing website, when users browse a recipe, the website can automatically categorize the recipe, and recommend other recipes of the same category.

There are lots of cuisines in the world, it is time-consuming and manpower consuming to categorize thousands of cuisines by human effort. The strategy is to categorize some cuisines by human effort, and then use supervised machine learning to build model based on the categorized cuisine data. After building model, program can use model to

automatically and quickly categorize cuisine whose category is unknown based on the ingredients of the cuisine. The following figure is the overall procedure of supervised machine learning:



Besides, the following figure is the overall step of project:



There are several published researches and various adoptable technique in this topic. For example, in [1], the author used extreme gradient boosting and random forest. In [2], the author use K nearest neighbor, support vector machine, and Naïve Bayes classification. Based on my past experience on machine learning, I would like to try random forest and gradient boosting decision trees, which are ensemble algorithms and are able to do feature selection. To compare performance between ensemble and non-ensemble algorithms, I chose single decision tree model as benchmark model.

The following are brief introduction of my adopted algorithms

- Decision Tree
  - There are two phases, training and testing phases. In training phase, it use training dataset to build a tree from a root node. In each node split, it selects best feature with cut point based on a goodness or error function. In classification phase, it classifies an instance based on its features and the tree model.
- Random Forest

    - There are two phases, training and testing phase. In training phase, it builds multiple decision tree models. In classification phase, each decision tree generates a result, and it will aggregate result of each decision tree to generate final result by majority voting.
- Gradient boosting decision tree
    - There are two phases training and testing phase. In training phase, it iteratively builds multiple decision trees. After it builds a single tree, the tree will be tested and calculated error, and the error will be used to build next decision tree. In classification phase, each decision tree generates a result, and it will aggregate result of each decision tree to generate final result by majority voting.

* Related researches:

[1] http://www.indjst.org/index.php/indjst/article/viewFile/106484/75562

[2] http://cs229.stanford.edu/proj2015/313_report.pdf

Problem Statement

Given a list of ingredient of a cuisine, program should automatically predicts the category of the cuisine. The dataset can be used by program is dataset of some categorized cuisines. Every cuisine can have different number of ingredients and different ingredients. Generally speaking, it is multi-class classification, where each category of a cuisine is a class.

Metrics

For multi-class classification problem, I think the most representative metric is accuracy, which is easy to compute and understand. As for precision, recall, and F1-score, their formula is simple in binary-class but complicated in multi-class, because each class can calculate an estimate value, and need to aggregate to generate an overall estimate. In aggregation, I chose “average with equal weight” to aggregate precision, recall, F1-score of each class, leveraging sklearn library to calculate.

Besides, for time-related metrics, I measure the elapsed time in training phase, and testing/validation phase.

This project uses the following metrics.

- Accuracy
  - $\text{Accuracy} = \frac{\text{number of instances which are correctly predicted}}{\text{number of all instances}}$
  - This project leveraged sklearn.metrics.accuracy_score to calculate.
- Average Precision
  - The topic is multi-class classification. For each class, calculate a Precision value of the class. And calculate average of Precision in all classes. (Each class has same weight, regardless of the number of instances of each class.)
  - $\text{Precision} = \frac{\text{number of true positive}}{\text{number of true positive + number of false positive}}$
  - This project leveraged sklearn.metrics.precision_score to calculate where argument “average” is “macro”.
- Average Recall
  - The topic is multi-class classification. For each class, calculate a Recall value of the class. And calculate average of Recall in all classes. (Each class has same weight, regardless of the number of instances of each class.)
  - $\text{Recall} = \frac{\text{number of true positive}}{\text{number of true positive + number of false negative}}$
  - This project leveraged sklearn.metrics.recall_score to calculate where argument “average” is “macro”.
- Average F1-Measure (Average F1-Score)
  - The topic is multi-class classification. For each class, calculate a F1-Measue value of the class. And calculate average of F1-Measue in all classes. (Each class has same weight).
  - $\text{F1} - \text{Measure} = \frac{2 \times \text{Prediction} \times \text{Recall}}{\text{Prediction + Recall}}$
  - This project leveraged sklearn.metrics.f1_score to calculate where argument “average” is “macro”.
- Training time
  - The elapsed time to train model, which used in training phase.
- Testing time
  - The elapsed time to test model, which used in testing or validation phase.

II. Analysis

Data Exploration

In Kaggle, there are two JSON files in this topic, train.json and test.json:

- The train.json contains around 40000 instances, each instance is data of a cuisine, containing cuisine id, cuisine label (the category of the cuisine), and a list of ingredient of the cuisine. The example of one instances:

```
{'cuisine': 'greek',
 'id': 10259,
 'ingredients': ['romaine lettuce',
                 'black olives',
                 'grape tomatoes',
                 'garlic',
                 'pepper',
                 'purple onion',
                 'seasoning',
                 'garbanzo beans',
                 'feta cheese crumbles']}
```

- The test.json contains around 10000 instances, each instance is data of a cuisine, containing cuisine id, and a list of ingredient of the cuisine (There is no label in test.json.)
- The difference between train.json and test.json is that train.json has labels, but test.json has no label.

To quickly get desired performance metrics, I decided to use part of train.json rather than test.jsona as testing dataset. After preprocessing train,json, I split dataset from train.json into training dataset (80%, 31818 instances), validation dataset (10%, 3978 instances), and testing dataset (10%, 3978 instances).

Exploratory Visualization

The following table are preprocessed data which are head 10 instances of dataset for training and validation. (Most columns are not displayed because of screen size) In modeling, column “category” is Y, the other columns are X. In the table, we can see some popular ingredients, such as pepper, and garlic. In observation, for an instance, values of most features are zero.

	category	romaine lettuce	black olives	grape tomatoes	garlic	pepper	purple onion	seasoning	garbanzo beans	feta cheese crumbles	...	Oscar Mayer Cotto Salami	Challenge Butter	orange glaze	cholesterol free egg substitute	ciabatta loaf
28889	italian	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
20857	chinese	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
21907	indian	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
28667	mexican	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
25618	indian	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2730	mexican	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
29572	southern_us	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0
377	french	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
23575	mexican	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2442	indian	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0

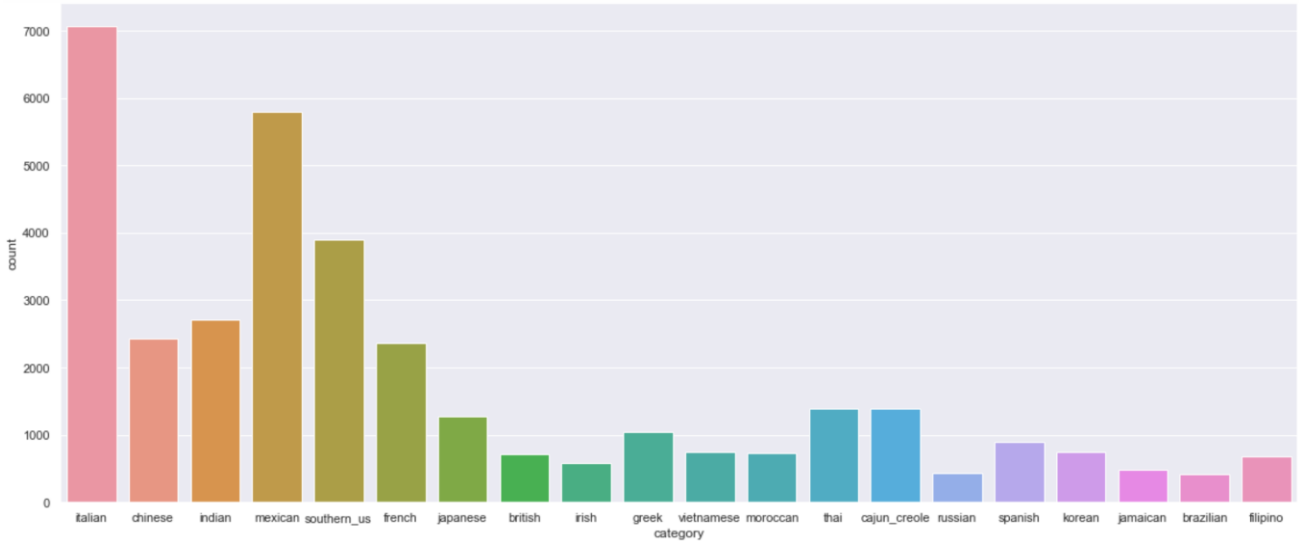
10 rows × 6715 columns

The following figure is some statistics of dataset for training and validation. (Most columns are not displayed because of screen size). In observation, there are 6714 features. The model should be able to do feature selection because there are so many features.

	romaine lettuce	black olives	grape tomatoes	garlic	pepper	purple onion	seasoning	garbanzo beans	feta cheese crumbles	plain flour
count	35796.000000	35796.000000	35796.000000	35796.000000	35796.000000	35796.000000	35796.000000	35796.000000	35796.000000	35796.000000
mean	0.006509	0.005587	0.005755	0.185943	0.111549	0.047296	0.003548	0.003632	0.008940	0.003632
std	0.080417	0.074540	0.075643	0.389066	0.314815	0.212274	0.059459	0.060155	0.094127	0.060155
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 6714 columns

The following figure are distribution of each category in the dataset which is for training and validation. Y axis is count of instances, and X axis is category. In observation, there are 20 categories.



Algorithms and Techniques

I adopted single decision tree as benchmark model, and selected Random Forest and Gradient Boosting Decision Trees (GBDT) for this topic. Single decision tree model is non-ensemble and has good ability in feature selection. Radom Forest and GBDT extend the decision tree to have ability to do feature selection. Besides , Random Forest and GBDT are ensemble algorithms which usually are more reliable than non-ensemble algorithms. The following are the introduction of my adopted algorithms and concept of ensemble learning:

- Decision Tree
    - There are two phase, training and testing.
    - In training phase, the algorithm build a tree based on training dataset. Each instance of training dataset contains feature values and a label. In beginning, all instances are placed in single root node, and the node will be split to generate its child nodes. When a node generates N child nodes, the algorithm will select a best feature with cut point to split the dataset of parent node into N subset based on a goodness function or error function. The algorithm keeps split its leaf node until it reaches stop criteria, such as all instances of each leaf node has same label. After the tree is build, each leaf node will generate a label by aggregating the labels of its own instances.

    - In classification phase, the algorithm is to classify an instance based on the features values
- Random Forest
    - There are two phase, training and testing.
    - In training phase, the algorithm builds multiple decision tree, and each tree is built by a subset of origin training dataset by random sampling with replacement. Besides, in each node to split, it will choose best feature with cut point from a random subset of remaining features. Remaining features means the features which have not been used to split node between the node and root node.
    - In classification phase, each decision tree in random forest generates a classification result, and the algorithm aggregate the results to generate a final classification result.
    - Random Forest is a kind of ensemble method, and has the capability to do feature selection
- Gradient Boosting Decision Trees (GBDT)
    - There are two phases, training and testing.
    - In training phase, the algorithm iteratively build multiple decision trees. After a tree is built, the tree will be tested, and the testing error will be used to train next tree.
    - In classification phase, each decision tree in GBDT generates a classification result, and the algorithm aggregates the results to generate a final classification result.
- Ensemble learning
    - The concept of ensemble learning is to combine multiple base models which are a little bit strong into a very strong model. In ensemble learning, each base model should have higher accuracy than a random guess model, and each base model should be different from each other. The concept is like that the decision combined by the decisions of three people who have different point of view and do not make decision by random is more reliable than the decision made by a very smart person.

Benchmark

I adopted simple decision tree model as benchmark model with default parameter setting excluding random state. I would like to compare performance between ensemble method with non-ensemble method.

III. Methodology

Data Preprocessing

The topic is multi-class classification. I adopted supervised machine learning. In instinct, X is related to ingredient of cuisine, and Y is category of cuisine. In order to build model, I performed the following preprocessing:

- **Make each ingredient be a feature.** The value of the feature is zero if the cuisine does not use the ingredient, and the value of the feature is one if the cuisine uses the ingredient. For example, if a list of ingredient of a cuisine do not contain 'cream', the value of the feature 'cream' of this instance is zero
- Example of a cuisine:

id	elderflower syrup	wine	honeydew melon	unflavored gelatin	water	sugar	orange zest	· · ·	garlic
1119	1	1	1	1	1	1	1	· · ·	0

Implementation

The detail code is in cuisine_categorization.ipynb. I developed by Python 3.7 on Jupyter on Anaconda3. The implementation used skelarn, datetime, pandas, numpy, IPython, seaborn,

matplotlib, pprint, and json. They can be installed by “pip install”. For example, we can use “pip install datetime” to install datetime in command line.

The following are further information of libraries which used in implementation:

Data Processing:

- json
- numpy
- pandas

Core Classification Model:

- class to build decision tree: sklearn.tree.DecisionTreeClassifier
- class to build random forest: sklearn.ensemble.RandomForestClassifier
- class to build gradient boosting decision trees: sklearn.ensemble.GradientBoostingClassifier

Model Selection:

- class to do grid search: sklearn.model_selection.GridSearchCV

Metrics

- class to calculate accuracy: sklearn.metrics.accuracy_score
- class to calculate F1-score: sklearn.metrics.f1_score
- The argument of “average” is set to “macro”. From sklearn document, “macro” means “Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.”
- class to calculate recall: sklearn.metrics.recall_score
- The argument of “average” is set to “macro”.
- class to calculate precision: sklearn.metrics.precision_score
- The argument of “average” is set to “macro”.

Visualization

- IPython
- seaborn
- matplotlib.pyplot
- pprint

Time:

- datetime

Refinement

After I validated random forest and gradient boosting decision trees, I observed that random forest build model faster than gradient boosting decision trees. Therefore, I chose random forest to do grid search to refine model.

- In grid search, I set up two parameters “n_estimators”, which means number of trees to build random forest, and “criterion”, which means the criteria to select attribute and cut point during tree growing. The criteria adopted in gird search is accuracy.
- To get acceptable grid search time, possible values of “n_estimators” are [10, 20, 30, 40, 50], and possible values of “criterion” is [“gini”, “entropy”]. The result of grid search are n_estimators=50, criterion=gini (default criterion is also gini).

IV. Results

Model Evaluation and Validation

Hardware:
OS: Windows 10 64 bit.
CPU: Intel® Core™ i7-7500U CPU @ 2.70GHz 2.90 GHz
RAM: 16.0 GB

I did model selection between two models, random forest and gradient boosting decision trees. Based on the performance result, I chose random forest. Although I did not perform cross validation on model selection, which may cause the accuracy comparison is not so reliable, training time comparison is still reliable. (Number of estimators/base models in both model is 10). In the following table, Random Forest is much faster and has higher accuracy than Gradient Boosting Decision Tree. Therefore, I chose random forest as final model.

Model Name	Training Time (s)	Accuracy (%)
Decision Tree (Benchmark Model)	33.494376	60.06
Random Forest	10.4799	66.26
Gradient Boosting Decision Trees	431.6963	58.35

After I chose random forest as final model, I performed model tuning (grid search) on random forest. I directly used training dataset to do grid search. After grid search, I use validation dataset to do validate Random Forest with best parameters. Finally, I use testing dataset to validate Random Forest with best parameters.

Model	**Phase**	**Parameter**	**Accuracy**	**F1 Score**	**Average Precision**	**Average Recall**	**Training Time(s)**	**Validation or Testing Time(s)**
Decision Tree (Benchmark)	Validation	random_state is 0, others are default	0.6066	0.4806	0.5015	0.4685	33.4944	0.2144
Random Forest	Validation	random_state is 0, n_estimators is 10, others are default	0.6626	0.5316	0.6154	0.4984	10.4799	0.2756
Gradient Boosting Decision Trees	Validation	random_state is 0, n_estimators is 10, others are default.	0.5835	0.4837	0.7057	0.4179	431.6963	0.7039
Random Forest (Parameters	Validation	random_state is 0, n_estimators	0.7029	0.5882	0.7274	0.5358	49.5050	0.6736

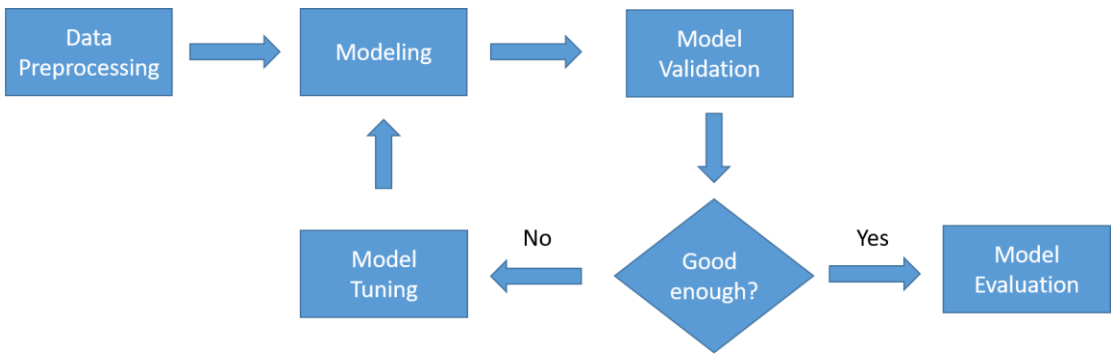
setting is from grid search)		is **50**, others are default.						
Decision Tree (Benchmark)	Testing	random_state is 0, others are default	0.6083	0.5006	0.5270	0.4840	33.4944	0.1719
Random Forest (Parameters setting is from grid search)	Testing	random_state is 0, n_estimators is **50**, others are default.	0.7159	0.6010	0.7413	0.5458	49.5050	0.7031

Justification

No matter on validation dataset or testing dataset, Random Forest achieved higher accuracy, higher F1 Score, higher average recall, and higher average precision than simple decision tree model (benchmark model). Although training time of Random Forest is higher than simple decision tree model, the cost is acceptable.

V. Conclusion

Wrap Up

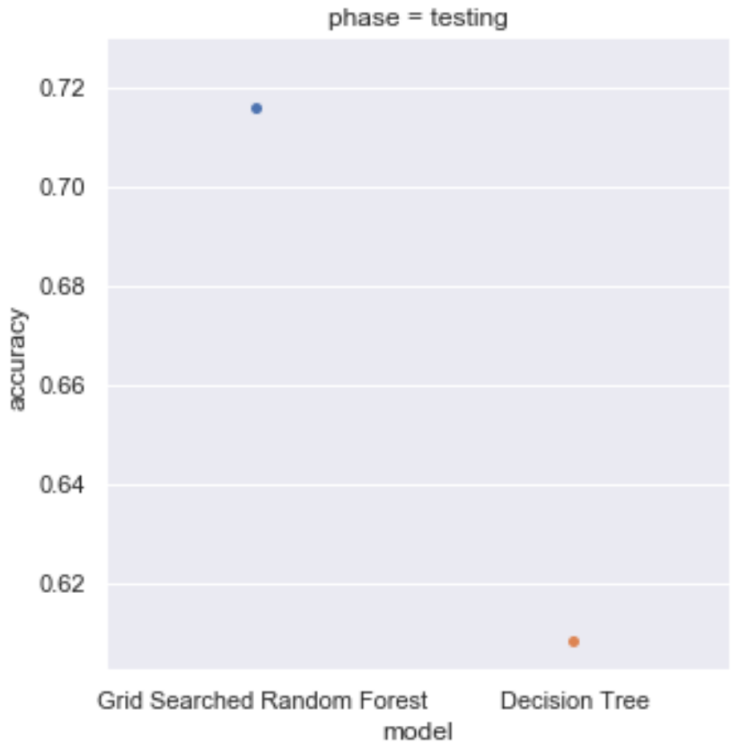


Key point in overall step

- In data preprocessing, convert the raw data to modeling data to apply supervised machine learning. Let dimension of features of each instance be the same.
- Choose random forest in model selection mainly based on the accuracy of validation dataset and training time of training dataset.
- Use grid search to do model tuning on random forest.
- Finally, random forest achieved 0.7159 accuracy among 20 classes on testing dataset containing 3978 instances.

Accuracy comparison

The Y of the following figure is the accuracy of final model (Random Forest) and benchmark model (Simple Decision Tree) based on the testing dataset.



Reflection

Based on the evaluation result, I think the model works. It is because the model achieved up to 70% accuracy among 20 classes, and it did not view the label of testing data until testing, and there was enough dataset to test (3978 instances).

The final model achieved higher accuracy than simple decision tree. I think it is because Random Forest is ensemble learning, which is proved to be able to reduce overfitting and enhance model performance.

Improvement

The following are possible improvement:

- **Generating result by aggregating the result made random forest and gradient boosting decision trees** in testing phase may lead to higher accuracy. It is because this two algorithm are different than each other, means they look through data by different point of view. Based on the concept of ensemble learning, this combination may lead to better performance.
- **Perform boarder grid search on Random Forest by validation dataset** to get better parameter setting. Parameters of algorithm influences performance of algorithm. As long as we do not select best parameter setting based on testing dataset, we perform boarder gird search may increase the testing accuracy without overfitting concern.
- **Try Extremely Randomized Trees.** Extremely Randomized Trees is similar to Random Forest. Extremely Randomized Trees has more variety between based model by selecting a cut-point at random in training phase, which may reduce overfitting and lead to higher accuracy than Radom Forest in some dataset.