Machine Learning Engineer Nanodegree

Capstone Project

Che-Chuan Li Udacity
January 16th, 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 domain is cooking.
- The goal is to use a list of ingredients of a cuisine to predict the category of cuisine. Ingredients are such like pepper, grape tomatoes, etc., and categories of cuisine are such like Greek, Indian, etc.
- The adopted method is supervised machine learning.
- The dataset is from Kaggle.

Problem Statement

Given a list of ingredient of a cuisine, predict the category of the cuisine, such as Indian, Korean, etc.

Metrics

This project uses the following metrics.

- Accuracy
  - $\mathrm{Accuracy} = \frac{\text{number of instances which are correctly predicted}}{\text{number of all instances}}$
  - Have you provided reasonable justification for the metrics chosen based on the problem and solution?
- Average Prediction
  - The topic is multi-class classification. For each class, calculate a Prediction value of the class. And calculate average of Prediction in all classes (Each class has same weight).
  - $\mathrm{Prediction} = \frac{\text{number of true positive}}{\text{number of true positive} + \text{number of false positive}}$
- 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).
  - $\mathrm{Recall} = \frac{\text{number of true positive}}{\text{number of true positive} + \text{number of false negative}}$
- 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).
  - $\mathrm{F1 - Measure} = \frac{2 \times \mathrm{Prediction} \times \mathrm{Recall}}{\mathrm{Prediction} + \mathrm{Recall}}$
- Training time
  - The elapsed time to train model.
- Testing time
  - The elapsed time to test model.

II. Analysis

Data Exploration

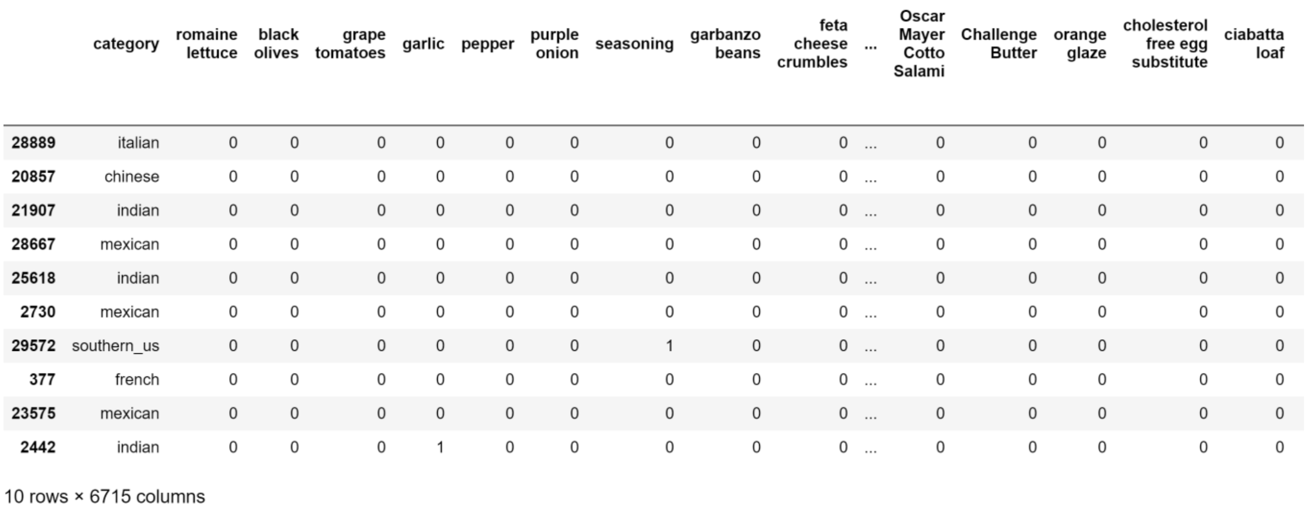
In Kaggle, there are two JSON files in this topic, train.json and test.json:

- 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

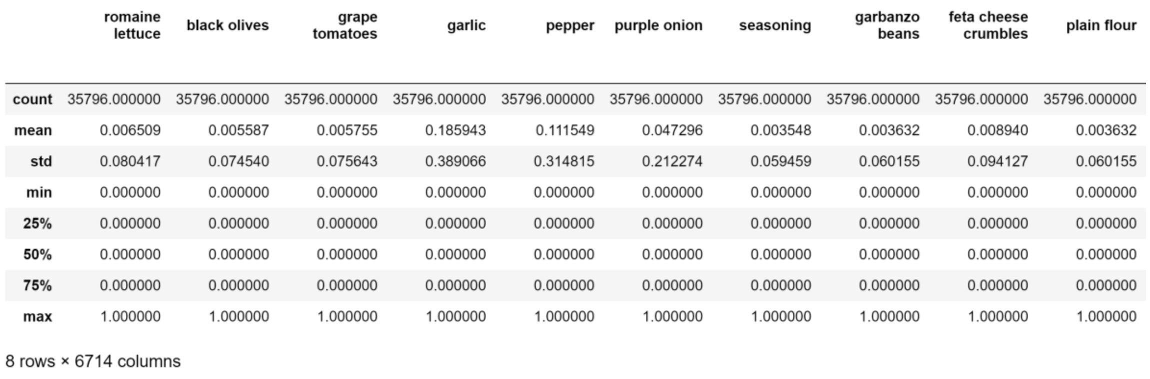
The following table are preprocessed data which are head 10 instances of dataset for training and validation. (Most columns are not displayed because of screen size) In modeling, column "category" is Y, the other columns are X. In the table, we can see some popular ingredients, such as pepper, and garlic.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

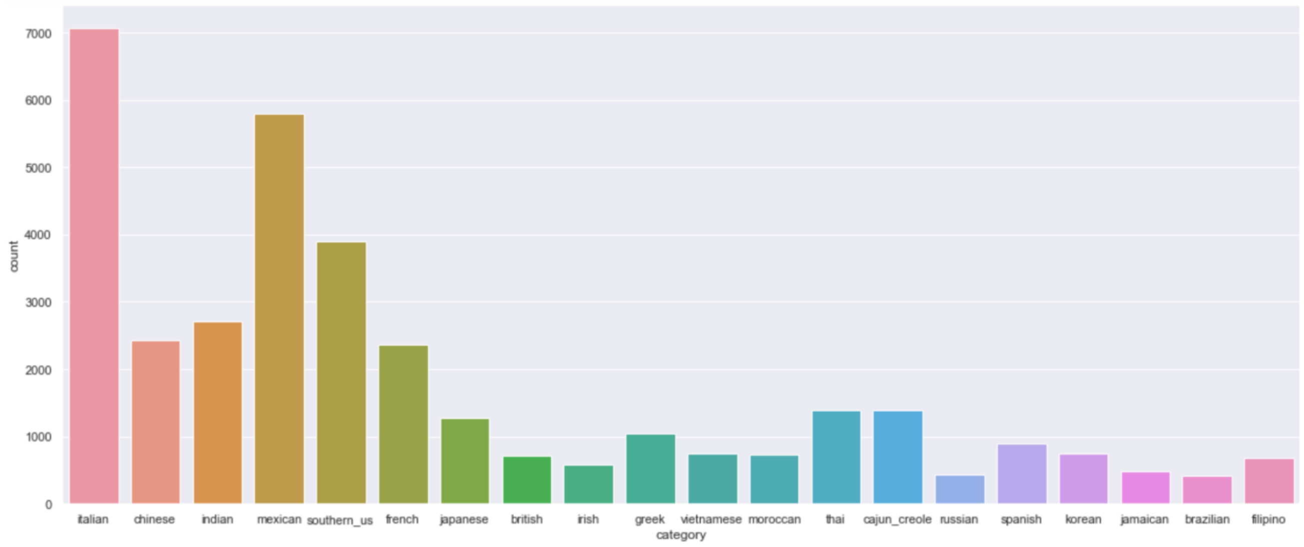
The following figure is some statistics of dataset for training and validation. (Most columns are not displayed because of screen size). In observation, there are 6714 features. The model should be able to do feature selection because there are so many features.



	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 ensemble supervised machine learning for this topic:

- Random Forest, which has the capability to do feature selection
- Gradient Boosting Decision Tree, which has the capability to do feature selection.

Benchmark

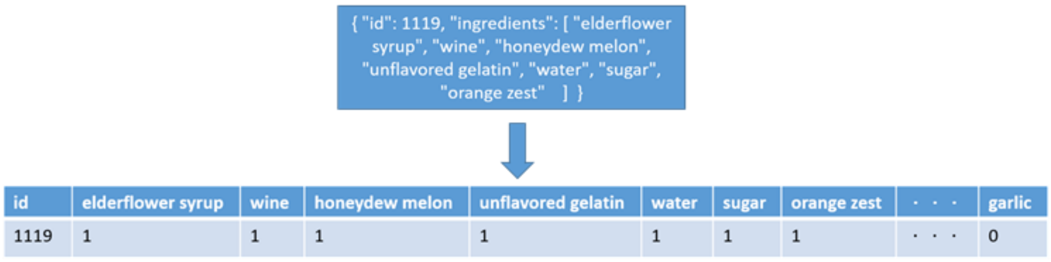
I adopted simple decision tree model as benchmark model with default parameter setting excluding random state. I would like to compare performance between ensemble method with non-ensemble method.

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": 1119, "ingredients": [ "elderflower
syrup", "wine", "honeydew melon",
"unflavored gelatin", "water", "sugar",
"orange zest"   ] }
```

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 followings are the main adopted libraries:

Modeling:

- 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
- 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

Data Processing:

- json
- numpy
- pandas

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.
- 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 setting is from grid search)	Validation	random_state is 0, n_estimators is **50**, others are default.	0.7029	0.5882	0.7274	0.5358	49.5050	0.6736
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

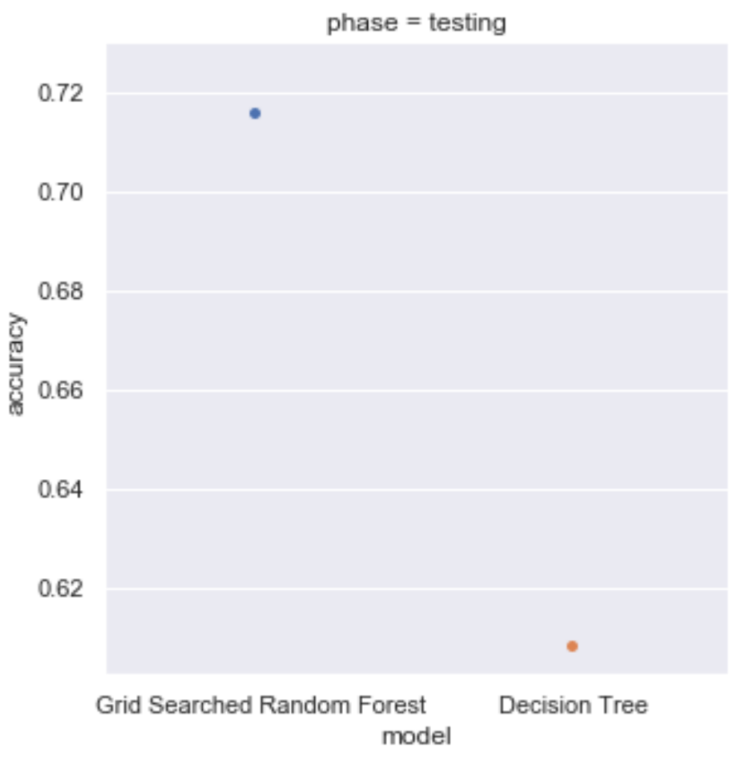
Justification

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

V. Conclusion

Free-Form Visualization

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

The final model achieved **0.7159** accuracy on testing dataset, which was not so good. Accuracy may be higher if the number of estimators (number of base model) in Radom Forest is higher.

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

- Increase the possible values of parameters during grid search on Random Forest.
- Try another model, such as Extra Trees, which is variant of Random Forest.