NCTU Data Mining

Group 13

1. Problem Statement:

Competition name: What's Cooking?
Duration: Wed 9 Sep 2015 – Sun 20 Dec 2015
Goal:

The goal of what's cooking is to help people find the cuisine of a dish based on the given ingredients. Hence, the competitor's goal is train the machine to predict the dish's cuisine with the training data provided by Yummy as shown as Figure 1. The competition started from September 9th 2015 and ended at December 20th 2015.

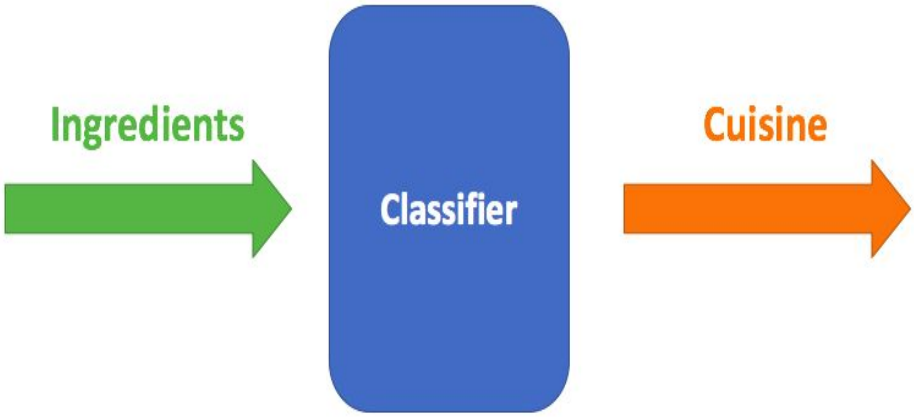


Figure 1. How classifier works

In the dataset, the official has given us a train.json file, a test.json file, and a sample submission csv file. In the dataset, it includes the recipe id, the type of cuisine, and the list of ingredients of each recipe (of variable length).

train.json - the training set containing recipes id, type of cuisine, and list of ingredients
test.json - the test set containing recipes id, and list of ingredients
sample_submission.csv - a sample submission file in the correct format

The data is stored in JSON format. The one training data input example is given as the following:

```
{
 "id": 24717,
 "cuisine": "indian",
 "ingredients": [
      "tumeric",
      "vegetable stock",
      "tomatoes",
      "garam masala",
      "naan",
      "red lentils",
      "red chili peppers",
      "onions",
      "spinach",
      "sweet potatoes"
 ]
 },
```


Train.json data number:39774
Test.json data number:9944

In the test file test.json, the format of a recipe is the same as train.json but without the cuisine type; the cuisine type becomes the target variable that we are going to predict. As for the submission, the submission file is a csv file that predicts the cuisine for each recipe in the test set. The file should contain a header and have the following format:

```
id, cuisine
35203, italian
17600, italian
35200, italian
…
etc.
```

2. Evaluation Criteria:

As mentioned in the problem statement, the rank and score is evaluated based on the categorization accuracy of the cuisine based on the given ingredients given by the test.json (the percent of dishes that you correctly classify).

3. Methodology and Tools:

For this project, we are coding with Python using Scikit tools and xgboost. The FIgure 2. showed the flow chart of our experiment, starting from understanding the the

training dataset to performing cross-validation of our prediction result.

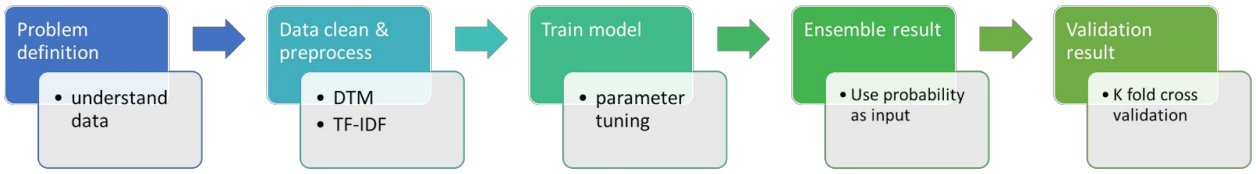


Figure 2. Experiment Flow Chart

Problem definition:

We carefully read the problem description, and then we loaded the training data into the program. We then scanned for any invalid input or anything odd with the ingredient so we could maximize the efficiency and accuracy when training the machine. We found that there is some superfluous words and redundant count words.(e.g. an apple, apples.)Though they appear as distinct ingredients, they are actually the same ingredient.

Data Cleaning & Preprocessing:

We use Scikit's natural language tool to perform word tokenization. It will remove punctations, stemming and lemmatizations (Ex: am, are, is → be. car, cars, car's, cars' --> car). We then use two techniques to transfer the clean data to a new feature space by DTM and TF-IDF. We also tune preprocessing parameters by using Scikit.

Training Model:

We trained with 7 classifiers; each model is executed with DTM data and TF-IDF data respectively. In addition, each classifier's parameters are tuned by Scikit, and we recorded the class label and class probability to determine which will be the best in this scenario.

Ensemble Result:

We use one single-layer and multiple-layer ensemble techniques (a.k.a stacked generalization) respectively. Though stacked generalization is theoretically way better, we cannot hit over 0.80 accuracy. Therefore, we used single-layer and have found that it can get 0.81 accuracy with class probability input.

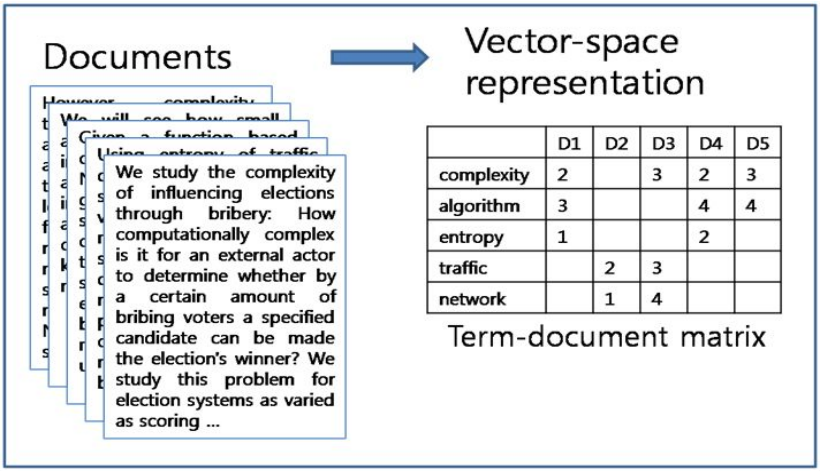
Validation Result:

To avoid overfitting and validation, we used 5 fold cross validation on the training dataset.

Now we will introduce the tools and techniques we used in the experiment. Some techniques are very commonly used in machine learning competitions. Grasping and understanding the tools is essential when applying them at each stage of the experiement.

Preprocessing parameter tuning:

Document Term Matrix(DTM): count the word in the document.



	D1	D2	D3	D4	D5
complexity	2		3	2	3
algorithm	3			4	4
entropy	1			2	
traffic		2	3		
network		1	4		

Figure 3. DTM Diagram [1]

Term Frequency–Inverse Document Frequency(TF-IDF):

reflect the term's value by its appearance frequency and distinctiveness

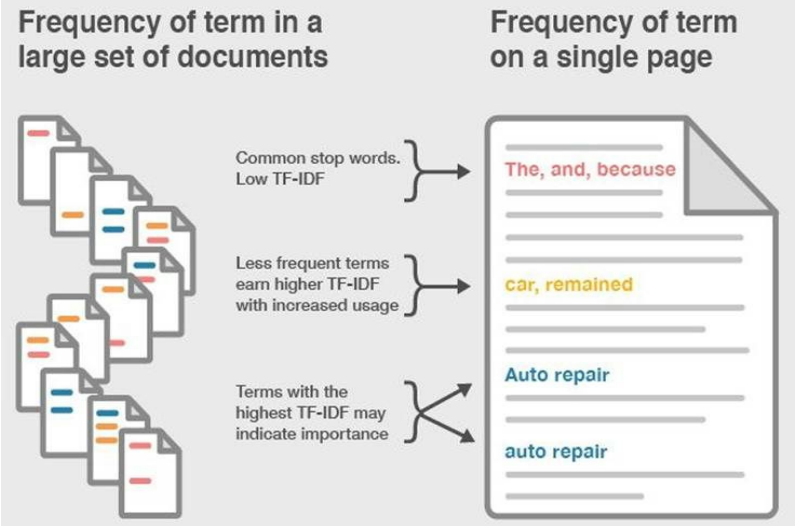


Figure 4. TF-IDF Diagram [2]

Use Scikit learn to tune preprocessing parameter

```
pipeline = Pipeline([
        ('vect', CountVectorizer(),
        ('tfidf', TFidfTransform()),
        ('clf', RandomForestClassifier()),
])
parameters = {
        'vect__max_df': [0.5, 0.7, 0.9],
```

```
        'vect__min_df': [0.01, 0.05, 0.1, 0.2],
        'vect__mx_features': [None, 2000, 4000],
        'vect__ngram)range': [(1,1), (1,2)], #unigrams or bigrams
        'vect__stop_words': [None, 'english'],
        'tfidf__use_idf': [True, False],
        'tfidf__norm': ['l1', 'l2'],
        'clf__n_estimators': [25]
}

grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1, verbose=1)
```

Classifier parameter tuning

GridSearchCV: exhaustively choose all parameter sets

RandomSearchCV: choose random parameter sets

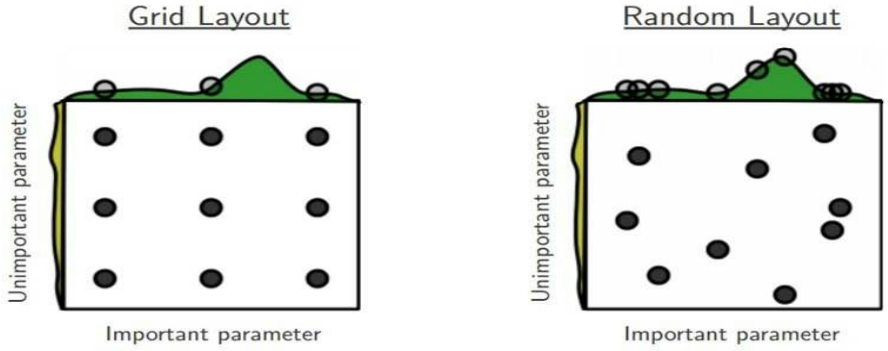


Figure 5. Grid and Random Search Diagram [3]

Classifier weight tuning

We manually tune each classifier's weighting. The thumb rule is weight is proportional to accuracy.

Classifier ensemble

We choose the classifiers with versatile based(Ex:tree-based, gradient-descent-based) and use class probability as input.

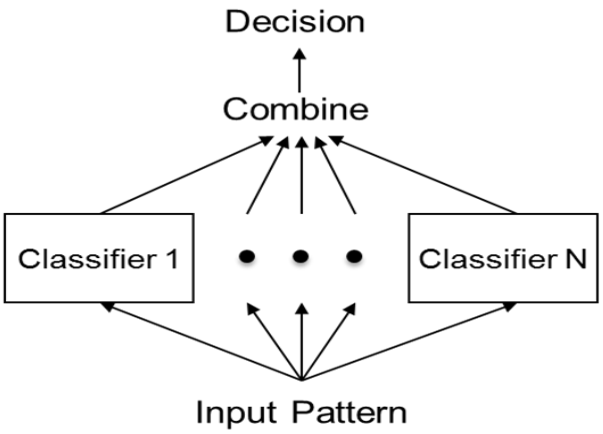


Figure 6. Ensemble Diagram [4]

Cross-validation

We take K turns, and each turn uses the K-1 split data to train and test on the 1 split data. Then after K turns, average K accuracy.

Figure 7. 5-fold Cross-validation Diagram [5]

4. Experimental Results:

Each classifier's parameter tuning and it's score :

- Random Forest
  - Mean validation score: 0.764 (std: 0.002)
  - Parameters:
    - 'bootstrap': False
    - 'min_samples_leaf': 1'
    - 'n_estimators': 275
    - 'min_samples_split': 3
    - 'criterion': 'gini'
    - 'max_features': 43
    - 'max_depth': None

- ExtraTree
    - Mean validation score: 0.765 (std: 0.003)
    - Parameters:
        - 'max_features': 6
        - 'n_estimators': 100
        - 'criterion': 'gini'
        - 'max_depth': None
- Stochastic Gradient Descent (SGD)
    - Mean validation score: 0.782 (std: 0.002)
    - Parameters:
        - 'penalty': 'elasticnet'
        - 'l1_ratio': 0.1
        - 'loss': 'modified_huber'
- Logistic Regression
    - Mean validation score: 0.786 (std: 0.001)
    - Parameters:
        - 'penalty': 'l2'
        - 'C': 8
        - 'tol': 0.01
        - 'dual': False
- Xgboost
    - Mean validation score: 0.802(std: 0.001)
    - Number of rounds: 2000
    - Parameters:
        - 'objective': 'multi:softprob'
        - 'eta': 0.09
        - 'max_depth': 5
        - 'num_class': 20
- K-Nearest Neighbour (KNN)
    - Mean validation score: 0.747 (std: 0.002)
    - Parameters:
        - 'n_neighbors': 16
        - 'weights': 'distance
        - 'leaf_size': 222
        - 'algorithm': 'kd_tree'
- MultinomialNB
    - Mean validation score: 0.734 (std: 0.001)
    - Parameters:
        - 'alpha': 0.05
        - 'fit_prior': True

Weight in ensemble:

When making a prediction, the classifier will generate the probability for each cuisine. We will add these values with weighting:

final_value(i) = sum(value(i, j) * weighting(j))

where **value(i,j)** means a classifier j will give a probability with respect to the data belongs to cuisine i , and **weighting(j)** means that the weighting of the value is given by classifier j. The final prediction will be the cuisine i that has the greatest final_value(i).

Table 1 shows our weight result for each classifier along with the individual prediction accuracy.

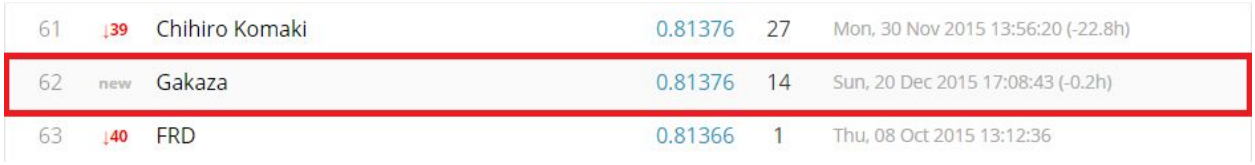
Table 1. Ensemble 7 algorithm (14 classifiers totally) and weight tuning result

Name	Raw accuracy	Raw Weighting	Tfidf accuracy	Tfidf Weighting
Random Forest	76.4%	1.5	75.9%	1.5
Extra Tree	76.5%	1.8	74.8%	1.8
SGD	76.3%	1.0	78.2%	1.0
Logistic Regression	78.1%	2.0	78.6%	2.0
XGboost	80.2%	3.0	80.7%	3.0
KNN	64.9%	0.5	74.5%	1.0
MultinomialNB	72.6%	1.0	73.4%	1.0

Final result :

Accuracy : 0.81376

Rank : 62/1388 (4.46%)



Total submission : 60

5. Conclusions and Lesson Learned:

In this project, we have learned how to automatically run the process by Scikit learn. Parameter tuning, model training , prediction ensembling and result validation all can be done automatically. We spent most time on parameter tuning , model training, determining which classifier weight combination work best in this competition and stacked generalization.

We have learned that how preprocessing like data cleaning and implementing efficient natural language processing plays an important role. The DTM and TF-IDF have made the raw data more reasonable. In addition, we can tune the parameter automatically with the help of Scikit learn.

We implemented 7 different models, and input with DTM and TF-IDF respectively, which generate 14 different outputs. We utilized Scikit learn to optimize the parameters automatically, and it took us a great deal of time.

The ensemble is a very powerful technique; it is quite accurate and stable. Some of the classifier like KNN or mutlinomialNB were thought to be useless because they had a relatively low prediction accuracy. However, they turned out to play a key role in ensemble even though their weight are low. The reason is that their algorithms beneath are very distinct and can discriminate data that others cannot do. Yet, KNN had only 0.5 weighting and multinomialNB had 1.0 weighting compared with the 3.0 weighting of xgboost.

Du to the shortage of time, we tuned the weight of classifiers in ensemble manually. Automatic weight calculation cost a lot of time and computation resources. Try-and-error is basically what we did to check whether an classifier should have higher or lower weighting, and we had attempted more than 60 times.

Actually, we have kept doing stacked generalization as figure 8 until last week. Stacked generalization appears to be better theoretically (e.g. Netflix use this technique), but we cannot exceed over 0.80 accuracy. It builds multiple layer of classifier and uses the result of each class' prediction from the lower layer as input to predict. Also, the team, whose

rank is number 6th in this competition, also used stacking in his implementation. So, we should give it a shot next time!

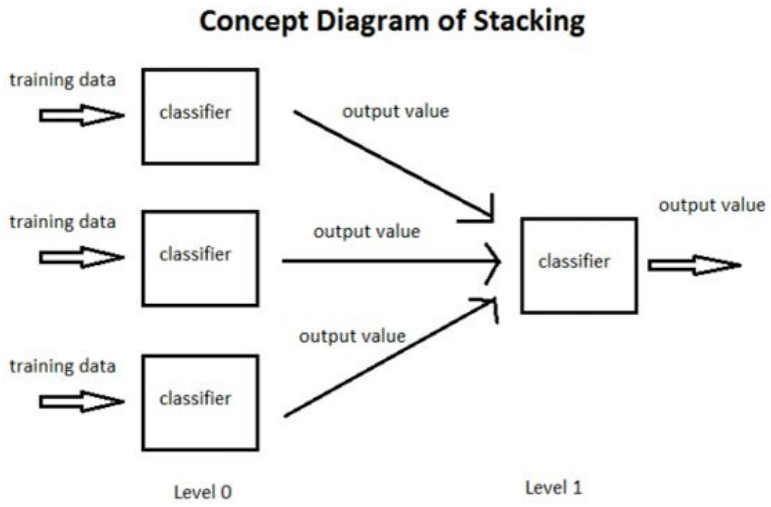


Figure 8. Stacking diagram [6]

Work Distribution:

0486016(潘亮宇): participate in all coding, algorithm design, task arrangement, presentation, Final report
0456151(張書晧): classifier coding & tuning, Final report
0386033(唐士傑): ensemble coding & tuning
0116205(邱一晉): preprocessing coding & tuning

Reference:

Xgboost : https://github.com/dmlc/xgboost
Scikit-learn : http://scikit-learn.org/stable

[1]. http://mlg.postech.ac.kr/research/nmf
[2]. http://www.googleseo.name/tag/tf-idf
[3]. https://medium.com/rants-on-machine-learning/smarter-parameter-sweeps-or-why-grid-search-is-plain-stupid-c17d97a0e881
[4]. http://mllab.sogang.ac.kr/?mid=research_subj_el

[5]. http://www.thefactmachine.com/logistic-regression
[6]. http://www.chioka.in/stacking-blending-and-stacked-generalization