Final Project on Kaggle Competition

What's Cooking

101021801 Yu-Hsuan Guan

NTHU Department of Mathematics

June 21, 2017

1. Introduction

The main problem

The main issue of this competition is to classify lists of ingredients into correct kinds of cuisines. There are only two given files `train.json` and `test.json`. Each training instance is represented in this format of JSON.

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

And, each testing instance is represented in the format of JSON.

```
{
    "id": 18009,
    "ingredients": [ "baking powder", "eggs", "all-purpose flour",
    "raisins", "milk", "white sugar" ]
}
```

There are totally 20 kinds of cuisines.

irish	mexican	chinese	filipino	vietnamese
moroccan	brazilian	japanese	british	greek
indian	jamaican	french	spanish	russian
cajun_creole	thai	southern_us	korean	italian

After an initial step of statistics, some basic summaries about data are given below.

Number of training instances	39774
Number of testing instances	9944
Total number of ingregients in training data	6714

The submission needs to be saved as a csv file of this form.

id	cuisine
18009	italian
35203	chinese
⋮	⋮

Obviously, this competition is a **supervised problem** of **multi-class classification**. There are several kinds of applicable classifiers.

- Transform to binary - **OvR** (one-against-all), **OvO** (one-against-one)
- Extend from binary - **Naïve Bayes**, **KNN** (IBk), **Decision trees** (J48), **SVM** (SMO), **Neural networks** (Multilayer Perceptron)

The main steps of learning

There are five major steps of machine learning in this project.

- **Data analysis** - We started with **data preprocessing** by these initial process.
    - Delete these special characters: ç, è, é, ®, and ™.
    - Convert all the strings into the type of UTF-8.
- **Visualization** - If we transform the training data into a sparse matrix full of 0 and 1 directly, the matrix will have the size of 39774×6714. To cope with this matrix more efficiently, we need **dimension reduction** to compress the size of matrix without losing too many varieties of data. **This step is the main difference between related work and our new method** which will be mentioned in the sections 2 & 3.
- **Modeling** - We chose **Weka** environment to create models. The detailed process of converting data matrix to the arff file for Weka environment will be mentioned also in the sections 2 & 3. All the codes used in this project can be found in the GitHub.
    - https://github.com/alicia6174/Kaggle-Whats-Cooking

    We skipped **tuning** in this project because we focused on comparing the results of two methods under different models.
- **Evaluation** - We presented various quantitiest to evaluate different models in the section 4. Finally, we gave the scores of our submissions on Kaggle site.
- **Prediction** - We saved the best model depending on the evaluation and used it to predict the cuisine of each testing data. The process will be mentioned also in the sections 2 & 3.

2. Related work

Descriptions of method

- **Dimension reduction** - The old method collected the **top ingredients** which occur most frequently in the training data as the features. To compare with our method, we chose the number of features to be 1000. In that way, each data could be transformed into a 1000-dimensional vector with the ith component being 1 if its ingredients contain the ith feature and being 0 if otherwise. The training data matrix of size 39774×1000 (without the header and labels) had this form and was saved as a csv file.

1	**2**	… …	**1000**	**cuisine**
0	0	… …	0	greek
1	0	… …	0	southern_us
⋮	⋮	… …	⋮	⋮
0	1	… …	1	mexican

- **Modeling** - We converted the csv file into an arff file so that the Weka environment would work more smoothly. We tried several multi-class classifiers for comparison. According to the evaluation (see §4), we saved the best model **SMO** to make predictions.

- **Prediction** - We repeated the steps of preprocessing and file conversion to create the needed file of testing data. The reduced testing data matrix of size 9944×1000 (without the header and labels) had the following form and was saved as a csv file. After converting it to an arff file, we used Weka again to predict the result. Finally, we saved the result as a needed submission file and uploaded it on the Kaggle site for scoring.

1	**2**	… …	**1000**	**cuisine**
0	0	… …	0	?
0	0	… …	0	?
⋮	⋮	… …	⋮	⋮
0	0	… …	0	?

Detailed steps

Codes & ML Tool	Created files	Goals
prefix_filter	train.json	delete special characters
create_top_ing.py	ing_top1000.csv	find top 1000 ingredietns
create_weka.py	train_weka_top1000.csv **(79.9M)**	create the reduced training data for modeling
weka-csv-arff.pl	train_weka_top1000.arff	convert to arff file
Weka		create models and make evaluations
Weka	train_weka_top1000_SMO.model	create the model of SMO
prefix_filter	test.json	delete special characters
create_weka.py	test_weka_top1000.csv **(19.9M)**	create the reduced testing data for prediction
weka-csv-arff.pl	test_weka_top1000.arff	convert to arff file
Weka	test_weka_top1000_SMO.txt	make predictions
weka-to-kaggle.pl	test_weka_top1000_SMO.csv	create the submission file for Kaggle

The 1001th attribute in the file test_weka_top1000.arff needs to be modified to the 20 cuisines before testing.

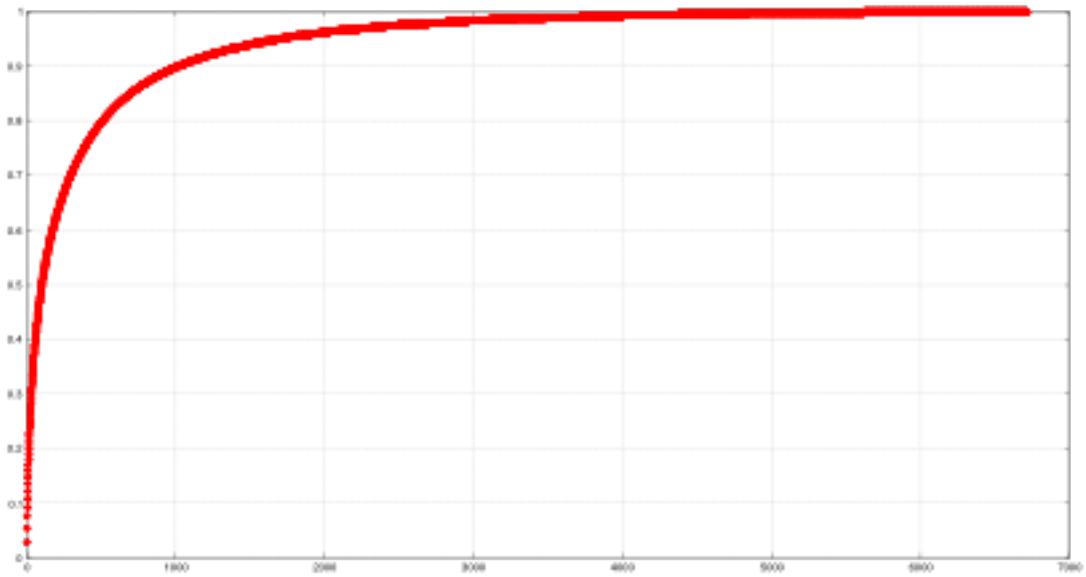
3. New methods

Descriptions of method

- **Dimension reduction** - Our method adopted **PCA** which is a linear unsupervised reduction. First we collected the totally 6714 ingredients as features and each data could be transformed into a 6714-dimensional vector with the ith component being 1 if its ingredients contain the ith feature and being 0 if otherwise. In that way, we could create the training data matrix of size 39774×6714. Second we computed the eigenvalues and eigenvectors of the corresponding covariance matrix. Third we chose the number of reduced dimension to be 1000 according to the score defined by

$$\text{Score}(k) = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{6714} \lambda_i}$$

  where λ_is are the eigenvalues which satisfy $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{6714}$. This grapf of score versus number of eigenvalues shows that 1000 corresponds to the score of 90.



Finally we multiplied the training data matrix by this matrix composed of the top 1000 eigenvectors to obtain the reduced training data matrix. Each feature had been normalized and rounded to the second decimal.

$$\begin{bmatrix} & & & \\ v_1 & v_2 & \cdots & v_{1000} \\ & & & \end{bmatrix}_{6714\times1000}$$

The reduced training data matrix of size 39774×1000 (without the header and labels) had this form and was saved as a csv file.

1	**2**	… …	**1000**	**cuisine**
0.71	0.34	… …	0.45	greek
0.49	0.57	… …	0.47	southern_us
⋮	⋮	… …	⋮	⋮
0.30	0.30	… …	0.47	mexican

- **Modeling** - This step was conducted almost in the same way as in the section 2. The main difference was that **SMO** still served as the best model after evaluation (see §4).

- **Prediction** - We conducted the same step of dimension reduction to obtain the reduced testing data matrix. The reduced testing data matrix of size 9944×1000 (without the header and labels) had this form and was saved as a csv file. The following steps was conducted in the same way as in the section 2.

1	2	… …	1000	cuisine
0.83	0.67	… …	0.52	?
0.93	0.63	… …	0.53	?
⋮	⋮	… …	⋮	⋮
0.70	0.20	… …	0.47	?

Detailed steps

Codes & ML Tool	Created files	Goals
prefix_filter	train.json	delete special characters
create_top_ing.py	ing.csv	find all the 6714 ingredients
create_mtx.py	train_mtx.csv	create the training data matrix of size 39774 x 6714
do_pca.cpp	eigVal_eiglVec	find the PCs and eigenvalues of the above matrix
create_eigVec.pl	eigVec	divide the file eigVal_eiglVec into eigVec and eigVal
create_eigVal.pl	eigVal	divide the file eigVal_eiglVec into eigVec and eigVal
create_pca_mtx.m	train_pca_mtx_1000.csv	create the reduced training data matrix of size 39774 x 1000 by matrix mutiplication
create_weka.py	train_weka_pca1000.csv **(187M)**	create the reduced training data for modeling
weka-csv-arff.pl	train_weka_pca1000.arff	convert to arff file
Weka		create models and make evaluations
Weka	train_weka_pca1000_SMO.model	create the model of SMO
prefix_filter	test.json	delete special characters
create_mtx.py	test_mtx.csv	create the testing data matrix of size 9944 x 6714
create_pca_mtx.m	test_pca_mtx_1000.csv	create the reduced testing data matrix of size 9944 x 1000 by matrix mutiplication
create_weka.py	test_weka_pca1000.csv **(48.7M)**	create the reduced testing data for prediction
weka-csv-arff.pl	test_weka_pca1000.arff	convert to arff file
Weka	test_weka_pca1000_SMO.txt	make predictions
weka-to-kaggle.pl	test_weka_pca1000_SMO.csv	create the submission file for Kaggle

The 1001th attribute in the file test_weka_pca1000.arff needed to be modified to the 20 cuisines before testing.

4. Comparison results

Evaluation

Our devices were the **Virtual Machine Instances** on **Google Cloud Platform**. We applied for 2 virtual machines of this size.

- OS: ubuntu16-04
- HDD: 20G
- CPU: 1 vCP
- RAM: 6.5 GB

We used the ML tool **Weka Environment** with the version shown below. In this project, we needed to modify the heap size to 4G. (The default size was 512M.)

- Weka Environment for Knowledge Analysis Version 3.8.0
- Java version "1.8.0_121"
- Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
- Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)

We split the training data into **66.0% for training and the remainder for testing**. We didn't compute the k-fold cross-validation since the training data was too large. The correctness and the running time are shown in the table below. The running includes the time taken to build model and the time to test model on test split.

66% Percentage Split Correctness, $\%$ **Running Time,** $sec.$		
Models	**Top-ing Method**	**PCA Method**
IBk (k=199)	27.0206 1778.31	36.3085 2422.41
Naïve Bayes	62.6414 86.26	37.2846 55.37
J48	63.5732 1353.21	40.0281 598.85
SMO	72.4543 2871.48	73.2382 2401.43

We also focused on the quantities - Kappa statistic, MAE, AUC, and confusion matrix.

- **Kappa statistic** K shows the difference between the classifier and stochastic classification, which is a decimal in $[0, 1]$. $K = 0$ means no difference while $K = 1$ represents the classifier is totally different from the stochastic classification. Generally speaking, K is proportional to AUC and correctness. Therefore, the closer K approaches 1 ($K \approx 1$), the better the result of the classifier is.
- **Mean absolute error** is the average of absoluate error.

$$\text{MAE} = \frac{\sum_{i=1}^{9944} |e_i|}{9944}$$

- **ROC Area** is the area under the ROC curve which is a decimal in $[0, 1]$. The closer AUC approaches 1 ($\text{AUC} \approx 1$), the better the result of the classifier is. For multi-class classification problem, we need to evaluate AUC for each class respectively.
- **Confusion Matrix** is the matrix defined by

  $$(i, j)\text{-entry} = \text{number of counts for actual class is } i\text{th class and predicted class is } j\text{th class.}$$

  Therefore, the more dominated the diagonal is, the better the result of the classifier is.

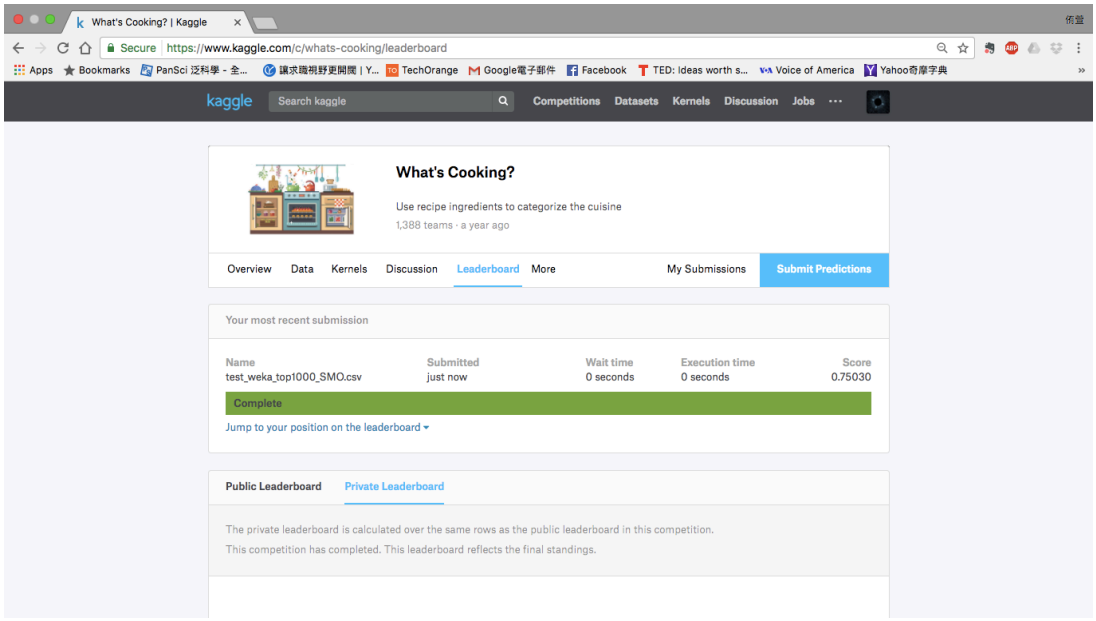
The Kappa statistic and MAE are shown in the following table. The results of AUC and confusion matrix can be found in the appendix.

Kappa statistic **Mean absolute error**		
Models	**Top-ing Method**	**PCA Method**
IBk (k=199)	0.1039 0.0804	0.1480 0.0884
Naïve Bayes	0.5861 0.0408	0.3340 0.0628
J48	0.5916 0.0440	0.3351 0.0607
SMO	0.6914 0.0905	0.7012 0.0905

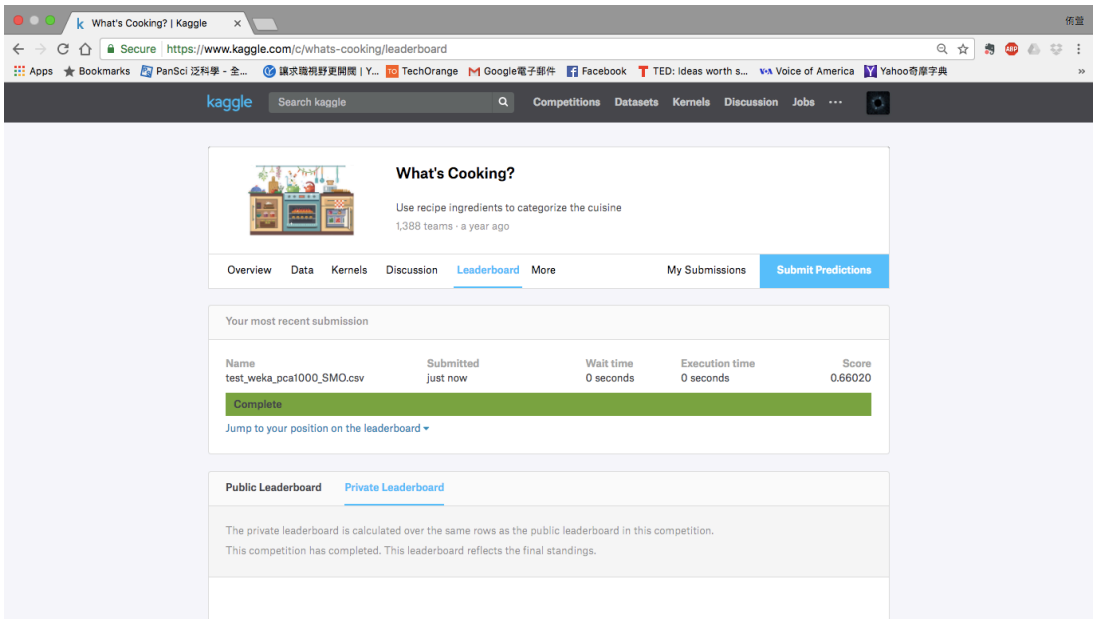
Kaggle score

The detailed process of testing are described in the sections 2 & 3. The followings are the final results.

- Top-ing Method **0.73994**



- PCA Method **0.66020**



5. Discussion and conclusion

We explain the reasons of some choices at first.

- We chose the number of features to be 1000 which has been explained in the section 3. On the other hand, we've tried 200 features for Top-ing method, but the result was not good. We've also tried 2000 features for PCA method, but our machines ran out of memory.
- We chose the models IBk, Naïve Bayes, J48, and SMO since they works for multi-class classification problem as we mentioned in the introduction. We've ever tried OvR, OvO, and Multilayer Perceptron. We didn't wait for the results because they all costed over one day.
- All the parameters in the models remained as the default except the number of nearest neighbors k in IBk. We chose $k = 199$ since the ideal value of k is the root square of the number of training instances.

$$\sqrt{39774} \approx 199.43$$

Now we discuss about the results of our experiments.

- The file size of the training data of PCA method (187M) is almost twice larger than the one of Top-ing method (79.9M). This result is due to the reason we rounded the data values of PCA method to the second decimal. This observation also holds for testing data (48.7M vs. 19.9M).
- The Top-ing method worked better than the PCA method. Why??? (Try to explain this in the aspect of the quantities in §4)

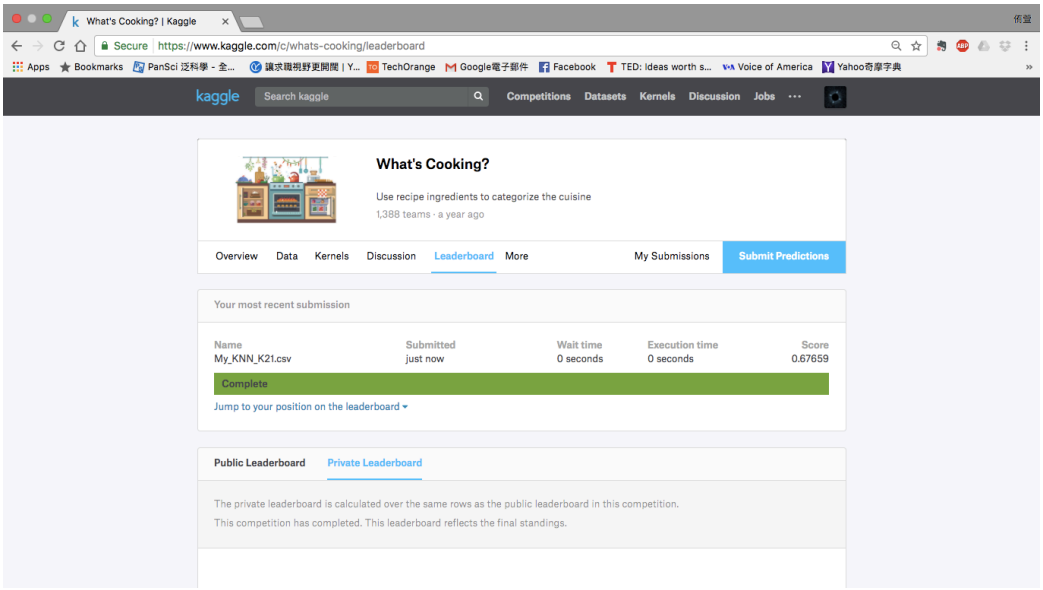
Finally, we end this project with some expectations and future work.

- We actually made a KNN algorithm ourselves with the distance defined by

  $$d(x_i, x_j) = \text{ number of different ingredients of } x_i \text{ and } x_j$$

  where x_i means the ith training data. We dealt with the raw data directly and chose $k = 21$ (chosen by tuning). This simple method created the better result than the one of PCA method.

  - My KNN **0.67659**

    You can find the code `My_KNN.py` also in the GitHub site.
  - https://github.com/alicia6174/Kaggle-Whats-Cooking

- This competition can be categorized as a **text mining** problem. The future work would concentrate on **Compressed Sensing**, **Hidden Markov Model**, and **Latent Dirichlet Allocation**.

References

1. 袁梅宇. *王者歸來：WEKA機器學習與大數據聖經 3/e.* 佳魁資訊, 台北市, 2016.