Machine Learning Kaggle Project

What's Cooking

101021801 Yu-Hsuan Guan

NTHU Department of Mathematics

June 28, 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**
Length of longest list of ingregients in training data	**65**
Length of shortest list of ingregients in training data	**1**
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, OvO
- Extend from binary - Naive Bayes, KNN, Decision trees, SVM, Neural networks, ELM
- Hierarchical classification

The main steps of learning

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

- **Data analysis** - We dealt with **data preprocessing** and decided the form of data representation in this step, which will be mentioned later in this subsection.
- **Visualization** - According to the above result, the data matrix derived from the training instances has size of 39774 x 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. The methods mentioned in the sections 2 and 3 adopted different ways to achieve dimension reduction.
- **Modeling** - We chose **Weka** environment to create models. The process of converting data matrix to the valid file for Weka environment will be mentioned in the sections 2 and 3. The detailed steps of how to use Weka will be described in the appendix.
- **Tuning** - In the section 3, we defined a specific score to determine the number of principal components used to reduce the dimension. And then, we used the **AutoWeka** tool to choose appropriate parameters needed in the candidate models. The detailed steps of how to use AutoWeka will be described in the appendix.
- **Evaluation** - In the section 4, we presented various quantitiest to evaluate different models. These

evidences showed that our method works better than the old one. Finally, we gave the scores of our submissions on Kaggle site.

Data preprocessing

- delete special characters(appendix)
- convert to unitcode(appendix)
- change to 01-vectors
- a matrix of size 39774 x 6714 (training) without missing values -> very large!

Detailed steps.

- prefixFilter create train.json → delete special characters
- :set ff=unix unicode
- a.encode('utf-8') unicode

2. Related work

- Top ing. -> sketch the method
- Dimension reduction -> Top ing.
- Modeling -> Weka (detailed steps)
- Tuning -> How to choose num. of top ing.?
- Testing
- Best: Top 1000, S=0.57; Top 1703, S=???
- We've tried Top 200 ing. + ing_len (normalized) -> Not good enough.
- The file size 81M vs. 187M due to the float type of PCA data.

Top 1000 ing. + ing_len (normalized).

- create*top*ing.py create ing_top200.csv
- create_mtx.py create train_weka_top200_len.csv (81M)
- ./weka-csv-arff.pl < ./train_weka_top200_len.csv > ./train_weka_top200_len.arff
- Weka (Percen66%)
- Naïve Bayes 63.3365 %
- IBk, k=1501 31.7533 %
- J48 64.2387 %
- SMO ??? %
- RandomTree 45.7739 %
- RandomForest Out of memory!
- MultiClassClassifier(OvR)Out of memory!
- MultiClassClassifier(OvO)Out of memory!
- Testing???

3. New methods

- PCA + SMO -> sketch the method
- Dimension reduction -> PCA, linear unsupervised reduction
- Modeling -> Auto Weka + Weka (detailed steps)
- Tuning -> 1. How to choose num. of PCs? Score.pdf 2. How to choose parameters of SMo? AutoWeka
- Testing
- Best: PCs 1000, S=0.66020; PCs 1703, S=???
- We've tried PCA 2000 (normalized) -> Out of memory while using Weka.

PCA 1000 (normalized).

- do_pca.cpp Have checked that eigenvectors are o.n.
- divide*into*vec_val.pl
- create*pca*mtx.m create train_pca_mtx_K1000_n.csv (normalized)
- create_weka.py create train_weka_tol1000_n_pca.csv (187M by round)
- ./weka-csv-arff.pl < ./train*weka*tol1000*n*pca.csv > ./train*weka*tol1000*n*pca.arff
- Weka (Percen66%)
- Naïve Bayes 3x.xxx %
- IBk, k=1501 30.8364 %
- J48 40.0281 %
- SMO 73.2382 %
- RandomTree 23.8335 %
- RandomForest
- MultiClassClassifier(OvR)66.2797 %
- MultiClassClassifier(OvO)
- Weka create train*weka*tol1000*n*pca_SMO.model
- create*mtx.py create test*mtx.csv
- create*pca*mtx.m create test*pca*mtx*K1000*n.csv
- create*weka.py create test*weka*tol1000*n_pca.csv (48.7M)
- ./weka-csv-arff.pl < ./test*weka*tol1000*n*pca.csv > ./test*weka*tol1000*n*pca.arff
- change the last attribute to cuisines
- Weka create test*weka*tol1000*n*pca_SMO.txt
- ./weka-to-kaggle.pl < ./ test*weka*tol1000*n*pca*SMO.txt > ./test*weka*tol1000*n*pca*SMO_sub.csv
- Kaggle score: 0.66020

4. Comparison results

Evaluation

- Correctness / Accuracy / Error rate

- K-fold cross-validation
- Confusion matrix
- ROC curve
- AUC value

Kaggle score

- Kaggle score (Screen Shot)

5. Discussion and conclusion

- Why did the new methods work better?
- Future work: Text mining, Compressed sensing, Factorization Machines (2010), Latent Dirichlet Allocation.

Appendix

How to use Weka & Version of Weka

Sketch the steps.

- Transform training data to a csv file.
- Try several multi-class classifications and choose features (ex. #(ing.) for each cuisine).
- Compute the accuracy and cross validation.
- Choose a model to test. ex. J48, KNN, PLA, Bayes...

Formal steps.

- Create a new csv data file (in a needed form).
- Convert it to UTF8 encoding. (use instruction in vim [A]).
- Convert it into train and test arff files (Hsin's shell-script [A]).
- Train train.arff by xxx on Weka, analyze the data (MAE, ROC…), and save xxx.model.
- Test test.arff by the model, and save result_xxx.txt.
- Convert result*xxx.txt to result*xxx.csv.

How to use Automatic Weka & & Version of AutoWeka

- For the newest Weka 3.8.0 and Auto-Weka 2.5, it needs to install Java and JDK and type the specific instruction to avoid java executable issue.

Coding

- You can find all the codes in this site. https://github.com/alicia6174/Kaggle-Whats-Cooking
- Please contact me if you have any question. yhguan8128@gmail.com

References

1. Y.-J. Lee, Y.-R. Yeh, and H.-K. Pao. Introduction to Support Vector Machines and Their Applications in Bankruptcy Prognosis. *Handbook of Computational Finance*, 731-761, 2012.

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