What's Cooking? Report

Prof. Wei Wang
TA: Wenchao Yu

Team Balloons
Haoying Wang (004-592-362)
Suphavit Pattanapitoon (404-303-751)
Zhiping Tan (204-021-272)
Wilsen Kosasih (604-501-335)

CS 145: Data Mining University of California, Los Angeles Fall 2015

TeamContribution

Haoying Wang: Weighted Voting, NBC, Error Correcting Code, Initial Coding, Report
Suphavit Pattanapitoon: Initial Coding, All Code Optimization, Report
Zhiping Tan: Weighted Voting, NBC, Decision Tree, Initial Coding, Report
Wilsen Kosasih: AvA, Weighted Voting, NBC, Initial Coding, Report

TableofContents

I. Introduction

What's Cooking? is an ongoing competition hosted by Kaggle and is accessible to the public. Participants are given two sets of data, one of which called the train dataset and the other called test dataset. The train dataset consists of lists of recipes, each of which consists of the id of the recipe, the type of the cuisine it belongs to, such as Chinese or Greek, and the list of ingredients used for the recipe. Meanwhile, the test dataset also provides a list of recipes, each with their own recipe id and list of the ingredients, missing only the type of cuisine the recipe belong to. The goal of this competition is then to urge the participants to classify correctly as much as possible which type of cuisine the recipes given in the test dataset actually belong to based on the data given on the train dataset. Thus, this is a classification problem.

II. BackgroundInformation

In Data Mining, classification is the process of predicting a certain outcome based on previous inputs. Usually, the input consists of a tuple of attributes which are already assigned to certain classes. The relationship between these tuples and the classes they are assigned to is then what is used to predict which class other tuples actually belong to. Classification is actually an important technique as it enables us to predict something simple such as the weather, or things much more complex such as whether a certain person is prone a certain disease. Additionally, there are so many different classification methods, which includes Support Vector Machines, Naive Bayesian Classifiers, and Decision Tree, each with their own strengths and weaknesses.

III. MethodofSolution

Programming Language: Java
Input Data Required: train.json, test.json

The first thing we did when we received the training data was finding out how many classes (or cuisine type) is possible in this problem. By parsing the recipes from the train data set, it was found that the recipes actually may belong to one of 20 different types of cuisine, which includes Greek, Southern US, Filipino, Indian, Jamaican, Spanish, Italian, Mexican, Chinese, British, Thai, Vietnamese, Cajun Creole, Brazilian, French, Japanese, Irish, Korean, Moroccan, and Russian. Parsing of JSON data is done by Google's Gson package. Since more than two classes exist, this is a multi class classification problem.

III.I. AllvsAllClassificationMethod

Concept

The first method we attempted using for solving the problem is the AvA classification method. AvA is a classifying method based on the Support Vector Machine or (SVM). By using a classifier, SVM transforms the original training data (or tuple) into a higher dimension, separating each attribute to belong to either of the two classes. When the test data is introduced, each attribute of the data is then tested. For each attribute of the data belong to the class, it gains one vote. In the end, the data is assigned to the class with the higher votes. However, SVM is designed for Binary Classification (Han, 2011).

Meanwhile, AvA is an alternative method that is usable for Multi class classification. The same as SVM, AvA separates which attribute belongs to each class. Then, when trying to classify the test data, it tests which classes each attribute belongs to. Every class that return a positive value receive an increase of one vote. In the end, the class with the highest vote is the one where the data is classified to (Han, 2011).

Algorithm&Explanation

For this problem, we can assume each of the 20 cuisine types to be a class. Each of the class has a set of attributes, which in this case is the ingredients of the recipe. The set of ingredients for each recipe is then what is used to classify the said recipe into one of the 20 available cuisine types.

To do this, first we identify all the possible ingredients that each type of cuisine can have. Then, for the test data, for each recipe, we check every cuisine type whether or not the ingredient exist for the class. If it does, increase its vote for the recipe by one. Otherwise, the vote stays the same. At the end of each recipe, the id of the recipe is then assigned to the class with the highest number of votes.

For a certain recipe with n ingredients, the vote gotten by class A would be

TotalVote(A) = Vote(ingredient a) + Vote(ingredient b) + … + Vote(ingredient n)
where,
Vote(ingredient i) = 1 if ingredient i is a possible ingredient in class A OR,
Vote(ingredient i) = 0 otherwise

Algorithm:
#Training Data Set

```
For each recipe in the training dataset
    Check the cuisine type the recipe belong to
    For each ingredient in the recipe
        Check if the said cuisine type already had the said ingredient
        If not
            Add the ingredient as a possible ingredient of the cuisine type
            Move to the next ingredient
        Otherwise
            Move to the next ingredient
    Move to the next recipe
```

#Test Data Set

```
For each recipe in the test dataset
    Set the vote for each class to be 0
    Save the recipe id
    For each ingredient in the recipe
        For each cuisine type possible
            Check if the ingredient is a possible ingredient of the class
            If yes
                Add vote for the class by 1
                Move to the next ingredient
```

Otherwise
Move to the next ingredient
Move to the next recipe
Find out which class has the most votes
Assign the recipe id to the class with most votes
Move to the next Recipe

By the end of the algorithm, each recipe id on the training dataset would be assigned to one of the 20 types of the cuisines. Also, because we did not use randomization in this algorithm, the accuracy rate of each try using this algorithm would stay the same.

Result&Analysis

When we uploaded the file containing the result from the algorithm, we received a **0.35288** accuracy rate, which is really low. This was to be expected, however, since the probability of several classes having the same vote would be quite high. This can be shown in the following example.

Assume:
From the training dataset,
-Cuisine A has {a,b,c,e,f} as possible ingredients.
-Cuisine B has {a,b,c,d,g} as possible ingredients.
Now, imagine there exist a recipe in the test dataset with {a,b,c} as its ingredients list. As a result, both Cuisine A and Cuisine B would have the same vote of 3. Because we did not use randomization on our algorithm, we would always choose Cuisine A to be the the class the recipe belongs to. If this happens to a lot of the recipes on the test dataset, the accuracy of the algorithm will fall down even more. Because of this, we feel that we need to weigh how much ingredient weight in each voting, giving more value to the ingredients that appeared more often in a particular class.

III.II. WeightedVoting

Concept

As stated above, to improve the accuracy of the classification, we felt that it is actually necessary to weigh how much each ingredient should carry for different type of cuisine. This is because there is a difference in how frequent each ingredient appeared on each class, so they should not be given equal amount of vote when it appears in a recipe. To differentiate it, based on the training dataset, we used the probability of an ingredient actually belonging to a certain class as the vote it actually acquires.

For a certain recipe with n ingredients, the vote gotten by class A would be

TotalVote(A) = Vote(ingredient a) + Vote(ingredient b) + … + Vote(ingredient n)
where,

Vote(ingredient i) = $\frac{Number\ of\ times\ the\ ingredient\ appeared\ on\ class\ A}{Number\ of\ times\ the\ ingredient\ appeared\ on\ the\ Training\ dataset}$ OR,

Vote(ingredient i) = 0 if ingredient i is not in class A

Algorithm&Explanation

The main algorithm for this method is mainly the same as the previous method. The only difference is only in the counting of the vote. Instead of adding each classes' vote by one every time an ingredient is present in the recipe, we add the ratio of the amount of time the ingredient appeared in the class to the amount of time the ingredient appeared overall into the vote instead. Because of this, we now had to also record how many times each ingredient appeared too. By doing this, we hoped that the difference of votes between all the classes would be widened so that we would be able to choose the most suitable cuisine type for each of the recipe in the test dataset.

Algorithm:
#Training Data Set

```
For each recipe in the training dataset
    Check the cuisine type the recipe belong to
    For each ingredient in the recipe
        Increment the total time the ingredient appeared on the training data set
        Check if the said cuisine type already had the said ingredient
        If not
            Add the ingredient as a possible ingredient of the cuisine type
            Set the amount of time the ingredient appeared on the class as 1
            Move to the next ingredient
        Otherwise
            Increment the # of time the ingredient appeared on the class
            Move to the next ingredient
    Move to the next recipe
```

#Test Data Set

```
For each recipe in the test dataset
    Set the vote for each class to be 0
    Save the recipe id
    For each ingredient in the recipe
        For each cuisine type possible
            Check if the ingredient is a possible ingredient of the class
```

If yes
Add vote for the class by x (x explained below)
Move to the next ingredient
Otherwise
Move to the next ingredient
Move to the next recipe
Find out which class has the most votes
Assign the recipe id to the class with most votes
Move to the next Recipe

$$x = \frac{Amount\ of\ time\ the\ ingredient\ appeared\ on\ the\ said\ class}{Amount\ of\ time\ the\ ingredient\ appeared\ on\ the\ training\ dataset}$$

Result&Analysis

When we uploaded the file containing the result from the algorithm, we received a **0.62359** accuracy rate. As can be seen, this is a huge improvement over the previous algorithm, with about 27% accuracy improvement or around double the accuracy. As expected, by differentiating how much vote each ingredient gets depending on the cuisine type, we were able to widened the differences between the total votes of each class and acquire better classification accuracy. As we did not use randomization in the algorithm, we receive the same result every submission. However, we felt that the accuracy is still pretty low. This might be because the way we count the vote is not exactly accurate as the probability will change depending on the training dataset. Higher amount of recipes in the dataset will lead to higher accuracy of the classification on the test dataset. Because of this, we decided to try NBC method to solve the problem.

III.III. NaiveBayesianClassifier(NBC)

Concept

NBC was the final classifying method we decided to use for this problem. NBC is a classifying method based on probability. Although the concept is simple, it has proven to be a very strong method in classification. It has advantage over the previous method in how it only requires a small amount of training data to obtain the parameters necessary to do classification. However, a problem may arise because Bayesian Theorem assume independence among attributes (Han, 2011).

For a certain recipe with n ingredients, the vote gotten by class A would be

TotalVote(A) = Vote(ingredient a) * Vote(ingredient b) * … * Vote(ingredient n) * Y
where,
Vote(ingredient i) = $\frac{Number\ of\ times\ the\ ingredient\ appeared\ on\ class\ A}{Total\ Number\ of\ recipe\ belonging\ to\ the\ class}$ OR,

Vote(ingredient i) = 0.000001 if ingredient i is not in class A

$$Y = \frac{Total\ Number\ of\ recipe\ belonging\ to\ the\ class}{Total\ number\ of\ recipe\ in\ the\ training\ dataset}$$

*Note that we did not multiply the vote by 0 when an ingredient is not in the class in order to avoid errors if there exist an ingredient in test dataset that did not exist in the training dataset, causing every class to get 0 votes.
The implementation this method is in `Cook.java`

Algorithm&Explanation

The main algorithm for this method is, again, mainly the same as the two previous methods, differing only in the way of counting of the vote. Here, we instead calculate the vote based on the NBC theorem. Because of this, we now had to also record how many recipes are actually in the training dataset. Additionally, we also had to find out which class each of the recipe belongs to and count them.

Algorithm
#Training Data Set

```
For each recipe in the training dataset
    Add the amount of total recipe by 1
    Check the cuisine type the recipe belong to
    Add the amount of total recipe belonging the type of cuisine by 1
    For each ingredient in the recipe
        Increment the total time the ingredient appeared on the training data set
        Check if the said cuisine type already had the said ingredient
        If not
            Add the ingredient as a possible ingredient of the cuisine type
            Set the amount of time the ingredient appeared on the class as 1
            Move to the next ingredient
        Otherwise
            Increment the # of time the ingredient appeared on the class
            Move to the next ingredient
    Move to the next recipe
```

#Test Data Set

```
For each recipe in the test dataset
    Set the vote of each class to be y (y explained below)
    Save the recipe id
    For each ingredient in the recipe
        For each cuisine type possible
            Check if the ingredient is a possible ingredient of the class
```

If yes

Multiply vote for the class by x (x explained below)

Move to the next ingredient

Otherwise

Multiply vote for the class by 0.000001

Move to the next ingredient

Move to the next recipe

Find out which class has the most votes

Assign the recipe id to the class with most votes

Move to the next Recipe

$$y = \frac{Amount\ of\ recipe\ in\ the\ training\ dataset\ belonging\ to\ the\ class}{Amount\ of\ total\ recipe\ in\ the\ training\ dataset}$$

$$x = \frac{Amount\ of\ time\ the\ ingredient\ appeared\ on\ the\ said\ class}{Amount\ of\ time\ the\ recipe\ in\ the\ training\ dataset\ belonging\ to\ the\ class}$$

Result&Analysis

When we uploaded the file containing the result from the algorithm, we received a **0.72647** accuracy rate. As can be seen, this is another improvement over the previous algorithm, with about 10% accuracy improvement on the result. This is because NBC does not require a lot of training data in order to classify correctly. However, as stated above, NBC also ignores dependency between attributes, or, in this case, ingredients. Yet, there do exist dependencies between ingredients in this problem. As an example, a recipe that requires baking sheet will also usually requires sugar and flour. Yet, NBC does not consider this example and treat them as 3 separate ingredients instead of 1. This is the reason why it would be hard to get higher precision by using NBC.

III.IV. OtherClassifyingMethods

DecisionTree [**QUI 93**]

Concept:

A decision tree represents a procedure for classifying data based on their attributes. The training data has only one simple attribute: ingredient. This makes it very hard to classify training data into sub-categories. We need to generate a decision tree with all the ingredients as branches. A little better idea will be combine frequent ingredients together to simplify the decision tree. But if a complete new ingredient in test data that doesn't appear in training data, we cannot make any prediction. In addition, we have 20 different cuisines. This will result in a very complex decision tree because we will have to make too many comparisons in each branch. Thus, we don't use decision tree to solve this problem.

Errorcorrectioncode [**Diet&Bak** 95]
Concept

Basically encode each cuisine in training data with 0,1; If certain ingredient appears, mark it as 1. If not, mark it as 0.

In test data, similarly, encode every unknown cuisine with 0 and 1. We can categorize each unknown cuisine based on Hamming distance (which is the number of different bits between two codes.) e.g. if the Hamming distance between unknown cuisine A and Indian food C is 2, and Hamming distance between unknown cuisine A and Chinese food A is 3. We should pick Indian food over Chinese food.

The problem with this method is that if the minimum Hamming distance is the same between unknown cuisine A, Indian food C and Chinese food A. We don't know which one to pick. Another problem is that if a complete new ingredient appears in test data, we can't even use the hamming code, which is based on solely training data.

IV. Conclusion

We chose NBC as our final algorithm because of its ability to classify in high precision even with small amount of training dataset. Despite ignoring dependencies between ingredients, it has proven itself to be fairly accurate in classifying the recipes with 0.72647 accuracy. Although other classifying method might have produced better result, their implementation would be really complicated due to the fact that there are 20 classes in this problem.

References

AvA:
Han, Jiawei, Micheline Kamber, and Jian Pei. Data Mining: Concepts and Techniques Concepts and Techniques. 3rd ed. San Francisco: Morgan Kaufmann In, 2011. 451-55. Print.

NBC:
Han, Jiawei, Micheline Kamber, and Jian Pei. Data Mining: Concepts and Techniques Concepts and Techniques. 3rd ed. San Francisco: Morgan Kaufmann In, 2011. 451-55. Print.

Decision Tree:
[QUI 93] QUINLAN J.R., C4.5: Programs for Machine Learning, Morgan Kaufmann, San Francisco, CA, 1993.

Error Correction Code:
[Diet&Bak 95] T. G. Dietterich and G. Bakiri (1995) "Solving Multiclass Learning Problems via Error-Correcting Output Codes", Volume 2, pages 263-286.