WHAT'S COOKING: PREDICTING CUISINES BASED ON INGREDIENTS OF RECIPES

He Huang, Ye Liu, Lichao Sun, Zhu Wang, Congying Xia, Fan Zhu
Department of Computer Science
The University of Illinois at Chicago
Chicago, IL, 60607, USA
{hehuang, yliu279, lsun29, zwang260, cxia8, fzhu20}@uic.edu

ABSTRACT

"There is no problem in the universe that delicacies cannot solve", said an ancient prophet. As mortal humans, we are all slaves of food, and some of us are so captivated by food that we will do anything to taste all possible cuisines on this planet and beyond. In this project, we explore the recipes of 20 different cuisines, and use several machine learning algorithms to predict the cuisine of each recipe based on its ingredients. We also compare the performance of these algorithms and discuss the factors that affect their performance.

1 INTRODUCTION

Here's a fun and delicious competition on Kaggle[1].

Picture yourself strolling through your local, open-air market...What do you see? What do you smell? What will you make for dinner tonight?

If you're in Northern California, you'll be walking past the inevitable bushels of leafy greens, spiked with dark purple kale and the bright pinks and yellows of chard. Across the world in South Korea, mounds of bright red kimchi greet you, while the smell of the sea draws your attention to squids squirming nearby. Indias market is perhaps the most colorful, awash in the rich hues and aromas of dozens of spices: turmeric, star anise, poppy seeds, and garam masala as far as the eye can see.

Some of our strongest geographic and cultural associations are tied to a region's local foods, but sometimes even the local people cannot distinguish some local dishes from exotic ones. Therefore, we try to design a machine learning model that can automatically recognize the category of a dish's cuisine given it ingredients.

1.1 DATA DESCRIPTION

We use the data distributed on Kaggle. The training data contain 39774 recipes that are stored in Json format, and each recipe contains three attributes:

- id: the cuisine ID
- cuisine: the category of cuisine that this recipe belongs to.
- ingredients: the ingredients of the recipe

There are 20 categories of cuisines, and 7137 different ingredients in total.

1.2 MACHINE LEARNING TASK DEFINITION

We formulate this task as a multi-class classification problem. Each recipe can be represented as a 7137-length vector, which can be used as the input of the machine learning algorithm. The output

[1] https://www.kaggle.com/c/whats-cooking

vector is the probability distribution of the recipes on 20 categories, where the probabilities sum to 1. Finally we pick the one with highest probability as our prediction.

The main evaluation metric we use is the overall prediction *Accuracy*.

1.3 Data Preparation

Although Kaggle provides a testing dataset, it has no ground-truth labels, and we can only submit our results 5 times per day to the Kaggle online judge to get the accuracy. To deal with this situation, we randomly sample 10% of the training data as our testing data, while maintaining the proportion of corresponding cuisines across two datasets as the same, so that the performance will not be significantly affected by different sampled testing data. The proportions of 20 categories of cuisines are shown in Table 1.

Table 1: Proportions of each category of cuisine

brazilian	british	cajun creole	chinese	filipino
1.2%	2.0%	3.9%	6.7%	1.9%
french	greek	indian	irish	italian
6.7%	3.0%	7.6%	1.7%	19.7%
jamaican	japanese	korean	mexican	moroccan
1.3%	3.6%	2.1%	16.2%	2.1%
russian	southern us	spanish	thai	vietnamese
1.2%	10.9%	2.5%	3.9%	2.1%

1.4 Data Cleaning and Preprocessing

All artwork must be neat, clean, and legible. Cooking is the supreme art, and so that the ingredients should be as neat, clean and legible as well. We first change all letters into lower cases, then remove digits and special characters, remove stop-words and do the stemming. After doing so, we calculate the Bag-of-Word (BoW) feature vectors as well as the TF-IDF feature vectors for each recipe.

Figure 1: Cooking is the supreme art of the universe

2 Data Exploration

Before diving into the magical black hole of machine learning, lets first have a look at some interesting things we discover in the dataset.

2.1 Histogram of the counts of cuisines

As is shown in Figure 2, we can see that Italian cuisine has the largest number of recipes, so there is no wonder why people say that Italy has some of the best food on this planet. Mexican food comes second, which may be due to the fact that Mexico shares much of its boundary with the United States. Affected by the Mexican foodies, people in southern U.S. also have a lot of ideas about food,

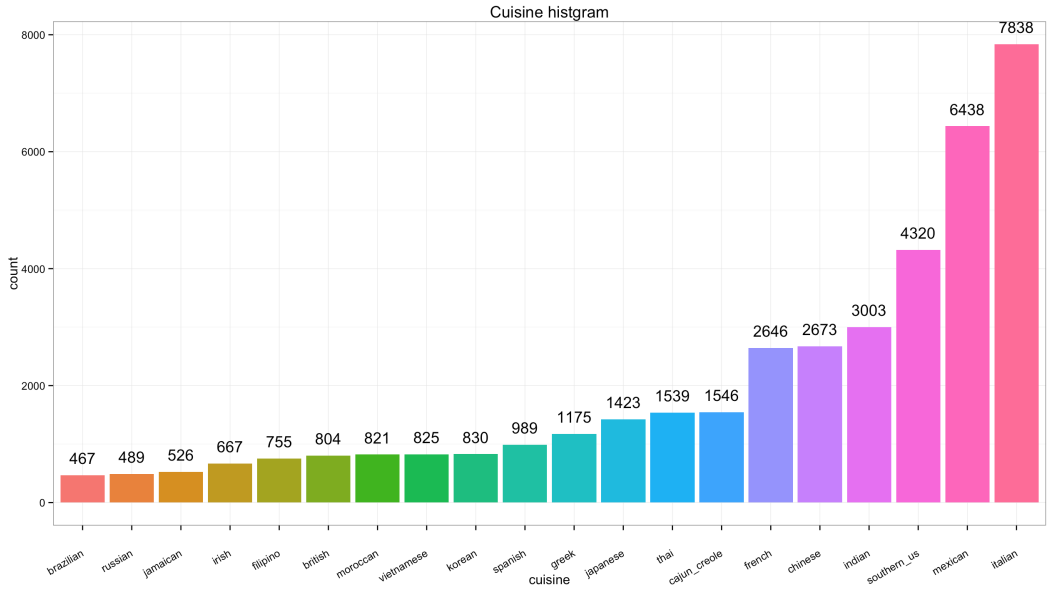

Figure 2: Histogram of the number of cuisines in each category

so we can see that southern U.S. cuisine has the third largest number of recipes. With the special spice and curry, as well as the large amount of immigrants to the U.S., Indian cuisine takes the forth place. Maybe because of the long distance, Chinese and French dishes only take the fifth and sixth places, but we should never underestimate the taste of Chinese and French food, since China and French are both praised as "The Country of Food".

2.2 Cuisine Clustering

We use TF-IDF feature vectors, and then apply Principle Component Analysis (PCA) on the dataset to reduce the features into 2-dimensional vectors. k-Means algorithm with $k = 5$ is performed to get the clusters. We visualize the results in a 2-dimensional space, where clusters are shown in different colors, and the sizes of circles are calculated by the Jaccard Similarity (one vs. the other cuisines in this cluster).

As we can see in Figure 3, Chinese, Japanese and Korean cuisines are grouped into the same cluster, which may represent the fact that they are all Asian countries and use similar ingredients in their dishes. Although Vietnam and Thailand are in a purple-color cluster different than the blue group, they are still very close to it, since they are still part of Asia.

A surprise is that the European countries split into two non-adjacent groups, with France and Italy leading one of them. We can easily understand that Russian and souther U.S. cuisines are in a group with British cuisine, since those countries have a long related history. However, it is hard to understand why Indian and Brazilian food is closer to the Italy group than other groups, and we should remember that India is an Asian country and Brazil is very very far from Italy.

2.3 Word Cloud

We plot the word cloud using the ingredients that have frequencies more than 1000, as is shown in Figure 4. The larger the word, the higher the frequency of that ingredient.

3 Machine Learning Algorithms

Now lets come to the fanciful machine learning part. We use five machine learning algorithms:

- Random Forest

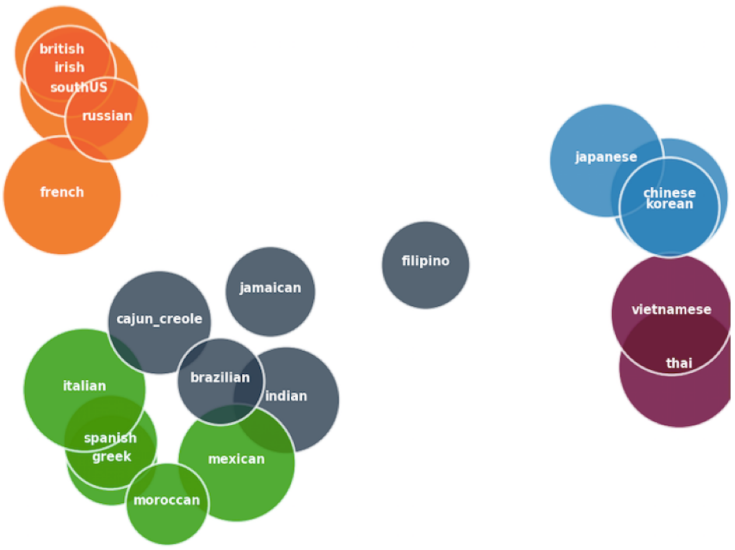

Figure 3: Visualization of cuisines after PCA and k-Means

Figure 4: Word cloud of ingredients

- Naive Bayes
- Support Vector Machine (SVM)
- Latent Dirichlet Allocation (LDA) & k-Nearest Neighbor (kNN)
- Neural Network

3.1 Random Forest

According to Wikipedia, random forests or random decision forests are an ensemble learning method for classification,regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean

prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over-fitting to their training set.

In our project, we use random forest for multi-class classification. We use "RandomForestClassifier function from the Python package *sklearn.ensemble* to build random forest classifiers. The most important part we have done in using random forest is parameter tuning.

We take four parameters into consideration. They are:

- $n\_estimators$. It represents the number of trees in the forest. When we increase the number of trees in the forests, the performance was improved, but the speed of the code becomes slower. So we set a series of $n\_estimators$ to do experiments to find out the optimal $n\_estimators$. This is the result of our experiment, shown in Figure 5

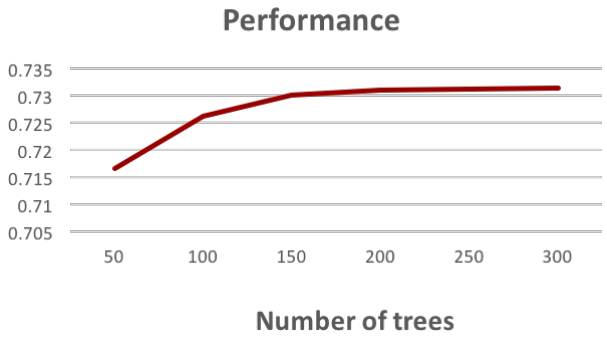

Figure 5: Accuracy w.r.t number of trees

  In order to obtain a good performance and a fast speed, we choose $n\_estimators$ = 200 as our optimal parameter.

- $max\_features$. The number of features to consider when looking for the best split. There are different types of max_features we can choose. The most frequent used tree types are:
  - $sqrt()$, then $max\_features = sqrt(n\_features)$
  - $log_2()$, then $max\_features = log_2(n\_features)$
  - None, then $max\_features = n\_feature$

  If the $max_features$ is too large, then the speed would be too slow. Also, reduce the maximum feature number it can have, avoids over-fitting. Empirical good default values are $max_features = n_features$ for regression problems, and $max_features = sqrt(n_features)$ for classification tasks $max_features$. In our experiments, we compare $sqrt()$ and $log_2()$. It turns out that $log_2()$ yields better performance. The results are shown in Figure 6.

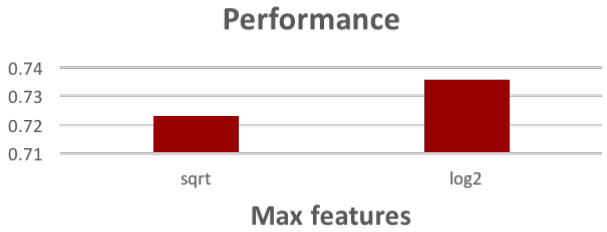

Figure 6: Accuracy w.r.t different $max\_featres$

- $Min\_samples\_leaf$. The minimum number of samples required to be at a leaf node. If int, then $considermin\_samples\_leafas$ the minimum number. The results are shown in Figure 7. In our project, we set $min\_samples\_leaf$ to 1.

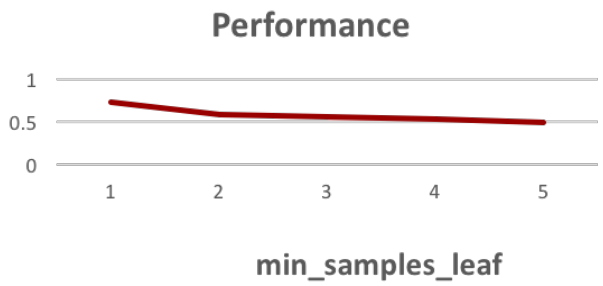

Figure 7: Accuracy w.r.t different $Min\_samples\_leaf$

- max_depth. The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than $min\_samples\_split$ samples. The results are shown in Figure 8.

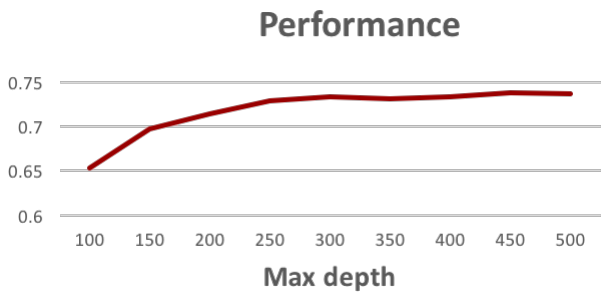

Figure 8: Accuracy w.r.t different max_depth

In this figure, it shows that add the depth of tree can improve the performance. In our case, since we only have 200 trees in the forest, we can just set this parameter as default and let nodes expanded until all leaves are pure or until all leaves contain less than $min\_samples\_split$ samples.

Overall, the best performance of Random forest is **73.7%**.

3.2 Naive Bayes

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.

Table 2: Results of Naive Bayes algorithm

	Accuracy	F1-score (micro)	F1-score (Macro)
Train	0.826601	0.826601	0.788229
Test	0.762946	0.762946	0.685015

The accuracy and F1-score of the results are shown in Table 2, and the ROC curve is shown in Figure 9.

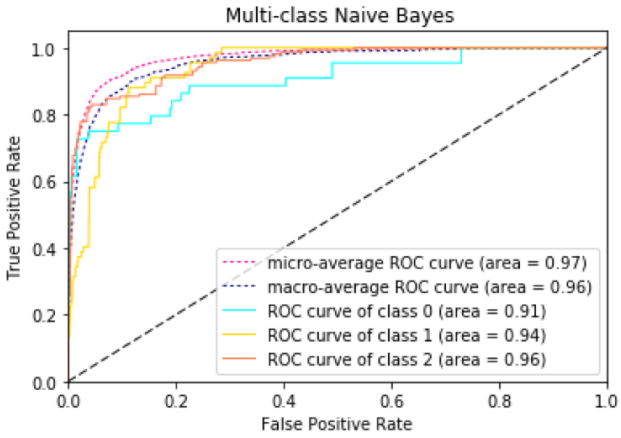

Figure 9: ROC curve of the results using Naive Bayes algorithm

3.3 Linear Support Vector Machine

Linear SVM is the fast machine learning algorithm for solving multi-class classification problems from ultra large data sets that implements an original proprietary version of a cutting plane algorithm for designing a linear support vector machine. LinearSVM is a linearly scalable routine meaning that it creates an SVM model in a CPU time which scales linearly with the size of the training data set.

In our project, we use Linear SVM (LSVM) for multi-class classification. We used "LinearSVC() from the Python package *sklearn.svm* to build multi-class classifiers. The most import part we did in random forest is parameter tuning.

- C: Penalty parameter of the error term, default is 1. The results are shown in Table 3.

Table 3: Accuracy of LSVM w.r.t. C

	Train	Test
0.5	0.922187	0.77382
1.0	0.933123	0.764395
2.0	0.940383	0.752325

- Loss: "hinge" or "squared_hinge", the results are shown in Table 4.

Table 4: Accuracy of LSVM w.r.t. different loss function

Loss	Train	Test
Hinge	0.908516	0.772567
Squared Hinge	0.933123	0.764395

As for the overall ROC, please refer to Figure 10.

3.4 Latent Dirichlet Allocation & k-Nearest Neighbor

3.5 Neural Network

4 Conclusions

"I always thought something was fundamentally wrong with the universe"[2].

[2]The Hitchhiker's Guide to the Galaxy

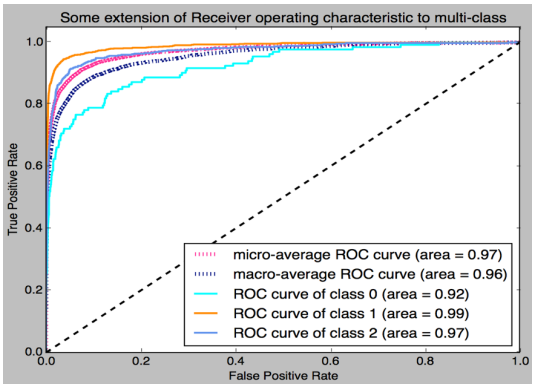

Figure 10: ROC curve of the results using Linear SVM

Figure 11: The Universe

ACKNOWLEDGMENTS

Use unnumbered third level headings for the acknowledgments. All acknowledgments, including those to funding agencies, go at the end of the paper.