Cuisine Identification
CIS 419 - Introduction to Machine Learning

Abstract

What should I make for dinner tonight? This is a universal question asked by home-cooks around the world as they peer into their fridges and pantries stocked with a diverse variety of ingredients. Indeed, these same chefs would find use for a system in which, given a list of ingredients, they could get a recommendation for what to cook that night. One good way to approach this timeless question is to view it through the lens of cuisine. By creating a cuisine classifier given a list of available ingredients, we will empower the everyday chef to expand his/her culinary boundaries into cuisines hitherto untouched and bring the tastes of faraway lands into the kitchens of all. As we often view the culture of other countries through their cuisine, this will also expose users of our classification system to new cultures and geographic regions.

We can easily see our work on this classifier being integrated into an app that recommends a cuisine of food, along with some common dishes of the region, given the ingredients that user has on hand. In addition, we believe that the commonly used ingredients of a region provide telling geographic and historical information about the culture and country, which could also be integrated into the app in order to educate and inform users of other cultures.

1. Introduction

1.1. Related Work

We regarded this task as a text classification with labeled training examples, and we examined ways that this problem has been tackled by previous research. Schneider [1] describes the Naive Bayes model, which often works well for text classification in practice. We noted that the multinomial NB model treats multiple occurrences of the same word in a document as independent, which is often not the case due to burstiness of word occurrences that we feel may be relevant to our data. This violated assumption will often cause the model to perform worse than the Bernoulli (binary) version, which is a consideration for our own model testing.

We also looked into the possibility of using support vector machines in text classification by looking at Joachimss paper [2] that detailed the usage of SVMs for learning text classifiers from examples, and noted that SVMs achieve substantial improvements over the currently best performing methods and behave robustly over a variety of different learning tasks.

Previous work related to using machine learning techniques to identify the type of cuisine of food was done by Zhang from the University of California San Diego [3]. Her work centered around using images to identify ingredients in a plate of food and then to classify these plates of food to the correct cuisine. Zhang implemented an SVM regression with RBF kernel to identify the ingredients in an image of a plate of food, and then an SVM classification with a third degree polynomial kernel to classify the plate of food to a cuisine label (based on the ingredients identified).

1.2. Data

The input data for this project originated from Yummly, an online recipe website that provided the dataset for a data prediction competition hosted by Kaggle. The particular competition we obtained this data from was called Whats Cooking? and seeks machine learning algorithms that can best use recipe ingredients to categorize cuisine. The data was provided in json format and included a recipe ID, the type of cuisine, and the list of ingredients of each recipe (of variable length). An example entry is shown below:

```
{
"id": 24717,
"cuisine": "indian",
"ingredients": [
"tumeric",
"vegetable stock",
"tomatoes",
"garam masala",
```

University of Pennsylvania, CIS 419/519 Course Project.

```
"naan",
"red lentils",
"red chili peppers",
"onions",
"spinach",
"sweet potatoes"
]
}
```

The data was split into two sets: training and testing data, the difference between the two being that the training data was labelled (contained the cuisine type) while the testing data was not. For our project, we decided to ignore the testing data provided by Kaggle (since without labels, there is no way to evaluate our models), and simply train and evaluate models on the training data with cross-validation.

Each recipe had a list of ingredients as text, so we examined text classification problems and methods used for this type of data. There are a total of 39774 recipes (i.e. instances) in the dataset.

1.3. Processing and Feature Selection

In our preliminary test, we converted the recipes directly to a sparse array of 39774 entries with 6714 unmodified ingredients, where each recipe entry would have indicators for which ingredients were present. Next, we examined the corpus created from the list of ingredients. To account for multiple cases of the same word, we changed all ingredients to lowercase only, and lemmatized the ingredients to remove inflections and keep only the base of the words; e.g. chopped and chopping both result in chop after lemmatization. We also used a tf-idf vectorizer to remove basic stopwords, as well as ingredients that occured in over half of the recipes (salt), since these carry less meaning than the other variables. This reduced the number of unique ingredients to 2962. We carried out separate tests that also removed words that occured too infrequently. Cutting words that appeared in less than .5% and .05% of recipes reduced the number to 418 and 1166, respectively. We also considered the multiple methods for grouping together comparable ingredients; e.g. organic milk, fat-free milk, and low-fat milk should all be the same ingredient. One idea was checking if any of the multi-word ingredients had the same word as any of the single-word ingredients, and if so, counting it as the single-word ingredient. However, we quickly found that this had the tendency to over-group the features (e.g. grape tomato vs. green tomato vs. green onion are generally used in different cuisines) and were thus losing both critical feature information and classification accuracy by using this preprocessing technique. We addressed brand names (e.g. eggland's best eggs) by checking ingredients for the registered trademark symbol and removing all words before it occurred.

2. Analysis

We tried multiple machine learning approaches that we thought might be effective in this context, as well as some less effective approaches that could be used for benchmarking performance. We used grid search (from SciKit) for parameter optimization. In the case of the decision trees and their related ensemble methods, parameters were tuned via 5-fold cross validation.

2.1. Decision Trees and Ensemble Methods

The decision tree classifier built-in with sklearn uses the Gini impurity as the criterion for evaluating splits to choose from at each node. The other option was to use Information Gain, however, a theoretical comparison of the two criteria done by Raileanu and Stoffel [4] found that they disagree only in 2 % of cases. The researchers used this as justification to state why in most published empirical studies, it was concluded that preferring one criterion over the other was impossible. The algorithm was specified to choose the best split at each node, with all features considered at each split. The parameter that needed tuning was the maximum depth of the decision tree. We evaluated trees with 1 level (stump), 10 levels, 100 levels, and 200 levels. We then computed an accuracy score for each tree, and found that the optimal depth was 200 levels.

2.2. AdaBoosted Decision Tree

We applied the AdaBoost ensemble method with a decision tree stump and 2-level decision tree. The boosting algorithms we chose (provided by sklearn) were the SAMME and SAMME.R boosting algorithm, which enable the Adaboost method to extend to multi-class classification cases [5]. The parameter to be tuned was the number of estimators (member classifiers) upon which boosting would terminate. In the interest of time, we tuned the parameter with only the AdaBoost-SAMME algorithm. The accuracy score was computed for each fold of the ensemble, and then averaged across all folds for 50, 100, and 200 estimators. We determined 200 was the optimal number of estimators because it was no longer time efficient to test AdaBoost-SAMME with higher numbers of estimators.

Table 1. Logistic Regression vs Naive Bayes vs SVM Accuracy

Lemmatization	LR	NB	Linear SVM
none	0.729	0.731	0.772
standard	0.777	0.715	0.778
sublinear term freq	0.774	0.739	0.774

Table 2. Decision Trees

Levels	Accuracy
1	0.201
10	0.384
100	0.581
200	0.596

2.3. Random Forest

Random Forest is a bagging ensemble method that constructs decision trees on bootstrapped replicas and aggregates their respective performances on out-of-bootstrap data. The Gini impurity was used as the function to measure the quality of each split (for reasons detailed above). Node decisions were restricted to the square root of the total number of features in order to increase diversity amongst trees. Increasing diversity reduces the bias introduced by the decision trees greediness (i.e. preference) for the best features. As with the AdaBoost ensembling method, we computed the average out-of-bootstrap accuracy score for each random forest ensemble through 50, 100, and 200 estimators, and similarly, the optimal number of estimators appeared to be around 200. Run time was not an issue, and it appears that accuracy simply maxes out around 200 estimators with random forest.

2.4. Naive Bayes

Naive Bayes is a probabilistic (generative) model that infers the most likely class of an unknown document using Bayes rule. Because estimating the joint probability distribution is not practical (as it severely overfits the data), the Naive Bayes classifier makes the (strong) assumption that all words in a document are independent. In the case of our data, the words can only show up in a recipe once, but the assumption is still faulty for the corpus. The Bernoulli and multinomial Naive Bayes models were quick for training and classifying, and the Bernoulli model predicted with accuracy comparable to the highest performing classifiers.

2.5. Logistic Regression

Logistic regression can be an effective classifier that determines a probability for output given input features. The multiple feature case uses a softmax function and is trained by maximizing the conditional likelihood of the training data. We used a multiple logistic regression which fit quickly and performed well, especially on the lemmatized data.

Table 3. AdaBoost vs Random Forest Accuracy

Estimators	AdaBoosted DS	RF
50	.303	.688
100	.337	.701
200	.394	.708

2.6. SVM

Support Vector Machine is a machine learning algorithm that can be effectively applied to text classification problems because of their ability to learn independently of the dimensionality of the feature space. Because text classification problems are generally linearly separable, SVMs, which function by determining linear separators with the smallest margin, are particularly effective for text classification problems such as this one. Joachims [2] outlines additional reasons that SVMs work well for text categorization. We used LinearSVC (a support vector classifier with a linear kernel) provided by SciKit to train on our data. We decided upon a linear SVM because it is better able to handle large sample sizes (such as in our case, with around 40,000 samples in total) than polynomial kernels are. Because we normalized the vectors, the linear kernel functioned the same as a cosine similarity kernel in this case, as per the SciKit documentation. For SVMs, features with high magnitudes tend to be biased, so feature scaling (normalizing our feature vectors) was a crucial step for getting this technique to be effective. In addition, we worked to optimize the parameters of the SVM by performing a grid search in order to arrive at the optimal value of C, which is the penalty for the error term.

3. Results

Evaluation and comparison of model performance: 5-fold cross-validation was used to tune model parameters for decision tree and related ensemble methods, while GridSearch was used to find optimal parameters for Logistic Regression, Naive Bayes, and Linear SVM. 5-fold cross-validation is a reliable metric for tuning because it directly tries to estimate a models performance on out-of-sample data.

We ended up compiling and comparing the results of various classifiers and preprocessing methods:

Preprocessing Methods

- Sparse array with parameter optimization
- Lemmatization
- Combining redundant features by brand elimination

Classifiers

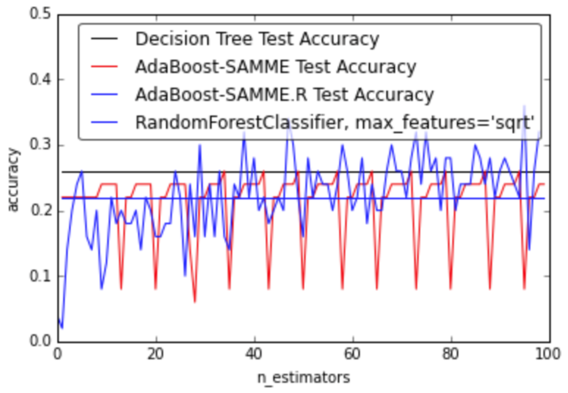
- Logistic Regression
- SVM

- Naive Bayes
- Decision trees and ensemble methods

Among the three classification systems we tried out, we found that the linear SVM consistently outperformed the other two both in terms of accuracy. Using sublinear term frequency in the vectorizer was able to improve Naive Bayes more, but we found our standard lemmatization process to produce the best results with the linear support vector classifier and logistic regression.

3.1. Final Performance Evaluation

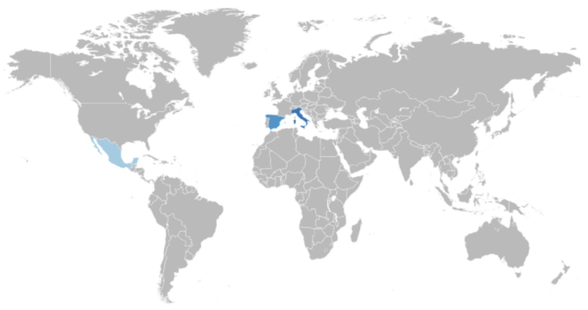
With our models tuned using 5-fold cross validation or GridSearch, we compare them head to head by training each on a randomly selected sample of 1% of the data (with replacement), and evaluated each models performance on hold-out set, that also consisted of a 1% of the data (with replacement), minus the data in the training set.



While the sample size is small, one is still able to discern that, on average, the Random Forest Classifier performed better than the rest of the decision tree classifiers and related ensemble methods.

3.2. Heat Map Display

In order to make our classifier more accessible and helpful to the everyday user, we implemented a heat map that indicates how likely it is that the ingredients come from a particular region of the world. We read in an svg file of a map of the world, drawn from Wikimedia Commons, and loaded into BeautifulSoup, a Python package for parsing XML documents, to map each country its location on the map. Then, using the predict_ proba method in SciKit, we output the probabilities that the recipe in question belonged to each of the different cuisines (classifications). Finally, we assigned a different color code to different ranges of probabilities, and updated the color scheme of the svg file for the countries that corresponded to the region of the probable cuisines. For example, the recipe used to generate the heat map above was an Italian recipe. The resulting heat map showed greatest likelihood of the cuisine coming from Italy (the darkest country), with Spain and Mexico being other possibilities. Of note, multiple other cuisines were predicted with negligible (defined as less than 5%) probability, and were not included on the heat map as they did not pass the probability threshold.



4. Conclusion and Future Work

We explored the classification of ingredient lists into cuisines. The linear support vector machine was the most effective at classifying the ingredient lists, with the Naive Bayes classifier coming in second and the decision trees classifier coming in a distant third.

While we analyzed a variety of potential methods for classifying the data (decision trees, naive Bayes, SVM), we could look into the effectiveness of using neural nets on this problem in the future. A paper by Ruiz [6] concludes that neural networks could be an important tool in text categorization, which suggests that neural networks might be effective in this context as well. Another classification technique that we would like to further look into is combining the classification and regression methods, perhaps by using a CART (classification and regression tree), as described by Luo [7].

We can see further uses of this classification system beyond the heat map visualization. One potential extension would be to get recipe rating data from Yummly using the recipe ID and use that to determine the expected rating of the dishes that could be made from the ingredients on hand. (For example, if you only had relish, sugar, and fish, the expected rating might be rather low as the only recipe that contains those ingredients is a nasty recipe for sweet relish fish.) Another potential extension would be to create an app that can have a dialogue with the user in order to determine what he/she could cook that evening. The app would brainstorm potential dishes and cuisines that could be made from the ingredients, as well as the needed ingredients to make said dishes, and then ask the user if he/she had the additional ingredients on hand. If not, it would move on the next most feasible cuisine or dish.

5. Acknowledgments

We would like to thank Professor Eaton for providing helpful suggestions for this project and, along with all the TAs, for providing this fascinating survey of machine learning methods and applications.

[1] Schneider, Karl-Michael. Techniques for Improving the Performance of Naive Bayes for Text Classification. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.101.6820

[2] Joachims, Thomas.Text Categorization with Support Vector Machines: Learning with Many Relevant Features. http://www.cs.cornell.edu/people/tj/publications/joachims_98a.pdf

[3] Zhang, Mabel Mengzi. Identifying the Cuisine of a Plate of Food. http://cseweb.ucsd.edu/classes/sp11/cse190-a/reports/mezhang.pdf

[4] Raileanu, Laura. Stoffel, Kilian. Theoretical Comparison between the Gini Index and Information Gain Criteria. http://link.springer.com/article/10.1023%2FB%3AAMAI.0000018580.96245.c6

[5] Zhu, Ji. Rosset, Saharon. Zou, Hui. Hastie, Trevor. Multi-class AdaBoost. https://web.stanford.edu/~hastie/Papers/samme.pdf

[6] Ruiz, Miguel. Automatic Text Categorization Using Neural Networks. http://courses.unt.edu/ruiz/Publications/asis-sigcr8.pdf

[7] Luo, Wei-yin. Classification and Regression Trees. http://www.stat.wisc.edu/~loh/treeprogs/guide/wires11.pdf

[8] Zhu, XiaoJin. Advanced NLP: Text Categorization with Logistic Regression. http://pages.cs.wisc.edu/~jerryzhu/cs838/LR.pdf