What’s Cooking?

CSE 5523

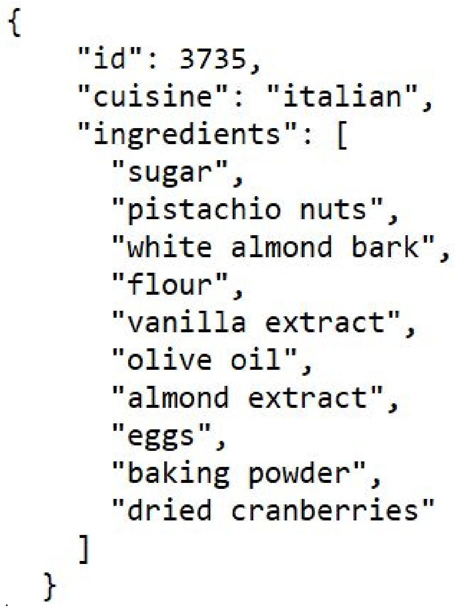
Project Code: github.com/noramyer/whats-cooking

Emily Engle, Frank Meszaros, Serena Davis, and Nora Myer

Background

The challenge for our chosen dataset was to classify a recipe’s cuisine given only its ingredients. The recipe dataset, which originated from the recipe recommendation website Yummly, is now freely available through Kaggle, and we thought that it would provide and fun and interesting text classification problem.

The recipes were represented as JSON objects with id, cuisine, and ingredients labels. “Ingredients” was an array containing anywhere from 4 - 20 unique ingredients that were part of the recipe. Below is an example input for a single recipe:



```
{
    "id": 3735,
    "cuisine": "italian",
    "ingredients": [
      "sugar",
      "pistachio nuts",
      "white almond bark",
      "flour",
      "vanilla extract",
      "olive oil",
      "almond extract",
      "eggs",
      "baking powder",
      "dried cranberries"
    ]
  }
```

Before processing the data into formats that would make it easy to classify, we collected some information on the recipes as a whole to have a better understanding of the breakdown of the recipes. There were a total of 39,774 recipes provided in a training file. The test set, however, lacked labels and we were unable to use that to test our accuracy unless we posted our work to Kaggle, so the training set was split to train and test our model. About 26,000 recipes were split into a training file called *train.json* while the remaining 14,000 are located in *test.json*.

There were a total of 20 cuisines and 6,714 unique ingredients represented. The top five occurring cuisines and ingredients can be found listed below.

Cuisine Type	Number of occurrences
Italian	7838
Mexican	6438
Southern US	4320
Indian	3003
Chinese	2673

Ingredient Type	Number of occurrences
Salt	18049
Olive oil	7972
Onions	7972
Water	7457
Garlic	7380

We also collected information on the top occurring ingredients for each cuisine, as well as cuisine and ingredient means, medians, and standard deviations. The complete analysis can be found in *data_analysis.txt*. This information provided us with a context of the dataset and insight into how we might best represent the data for classification.

Text Classification

Previously we have explored a few text classification problems such as email spam identification, news article topic classification, and movie review sentiment analysis. Though recipe classification is not quite a natural language problem in the traditional sense, it proved to be useful to approach our task that way since, like the datasets for other text classification problems, the recipe dataset is essentially made up of collections of words with associated labels.

A major difference between the cuisine classification task and other text classification tasks is that there is no need to consider grammar for cuisine classification. While many methods of text classification require the use of word order, parts of speech, and other grammatical considerations, these are all essentially meaningless for a list of ingredients. However, as we will see, the “bag-of-words” model, which disregards grammar and word order, works well for this dataset.

A particularly interesting similarity between this problem and other text classification problems is the drastic variation in relative importance of words. With the email spam identification problem, for example, common words like “the,” “a,” and “and” should essentially be ignored by the classifier, since those words are likely to be present in any type of email, spam or not. Similarly, for nearly every cuisine in our dataset, the number-one most common ingredient is “salt,” so the presence of “salt” in an ingredient list should mean almost nothing to a cuisine classifier. As we will discuss later, we expected the Term Frequency-Inverse Document Frequency (TF-IDF) statistic that is often used for text classification to be a useful tool to account for this issue of ingredients’ relative importance.

Data Representations

Feature Vector

Motivation
Much like a count vector, a feature vector indicates the presence or absence of a value in a matrix, where every row represents a different recipe. This is a common data representation in text classification because of the simplicity of its setup and use. It is a bag-of-words model where the multiplicity of every feature for a recipe cannot exceed one, since each recipe can have at most one count of a given ingredient. This data representation makes it very easy to calculate various measures to characterize the text, and it provides a good baseline to compare classifiers as well as compare other data representations.

Setup

After loading the data, the pre-processing step was the bottleneck of the text classification. Every recipe in both the training and test files was represented by a single row in a constructed feature vector matrix. Each column maps to a single ingredient found in one of the recipes and those are used as features during classification. Since there were nearly 6,000 unique ingredients, each recipe's feature vector was quite long. Below is an example of a single recipe's feature vector of length 5955:

[0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, . . . 0, 1, 0]

Results of Feature Vector Representation

After the recipes were encoded as feature vector matrices, three models learned this semester were used for analysis: Naive Bayes, Stochastic Gradient Descent (with three types of loss), and Logistic Regression. The accuracy from each resulting model is below.

Model	Loss	Accuracy (%)
Naive Bayes	N/A	72.37
Stochastic Gradient Descent	Hinge	75.51
	Log	69.77
	Perceptron	71.45
Logistic Regression	N/A	76.95

Term Frequency-Inverse Document Frequency (TF-IDF)

Motivation

One of the representations employed for this project was term frequency-inverse document frequency (also known as TF-IDF). The motivation for making use of this statistic was that our corpus had some heavily recurring terms such as “salt”, “onions”, and “olive oil” as well as others. The idea is that a term in the corpus is scored based on its frequency, but also would be multiplied by the inverse of how many different documents (or in this case, recipes) it appears in to better weight the score. Ideally, a more important term like turmeric would have a score greater than salt because salt shows up 11,629 times, across many different recipes, but turmeric is less likely to show up as frequently so it won’t have as a large of a denominator when compared to salt. Thus, turmeric would be scored more favorably when using this representation.

Setup

Fortunately after loading our data, pre-processing the data to be properly set up for TF-IDF was relatively painless. The idea was that each ingredient would be concatenated into a single string and our labels would have a numeric value assigned to them. The space between ingredients would be the delimiter between ingredients.

The biggest pain point with this representation was that our ingredients varied in terms of how many words were listed for a single ingredient. For example, some ingredients we saw were “sliced tomatoes” or “roma tomatoes”. Using scikit-learn, we needed to find a way to group the corpus in a way that made sense to us and also made sense to the scikit framework. Thus, the solution employed was to hyphenate multi-worded ingredients so as not to have a “sliced” ingredient and a “tomatoes” ingredient being separated as those two things are logically connected with respect to the ingredient.

Results of TF-IDF Representation

Our TF-IDF representation was tested against three models learned through the semester: Naive Bayes, Stochastic Gradient Descent (with three types of loss), and Logistic Regression.

Model	**Loss**	**Accuracy (%)**
Naive Bayes	N/A	63.07
Stochastic Gradient Descent	Hinge	71.96
	Log	74.52
	Perceptron	67.64
Logistic Regression	N/A	75.81

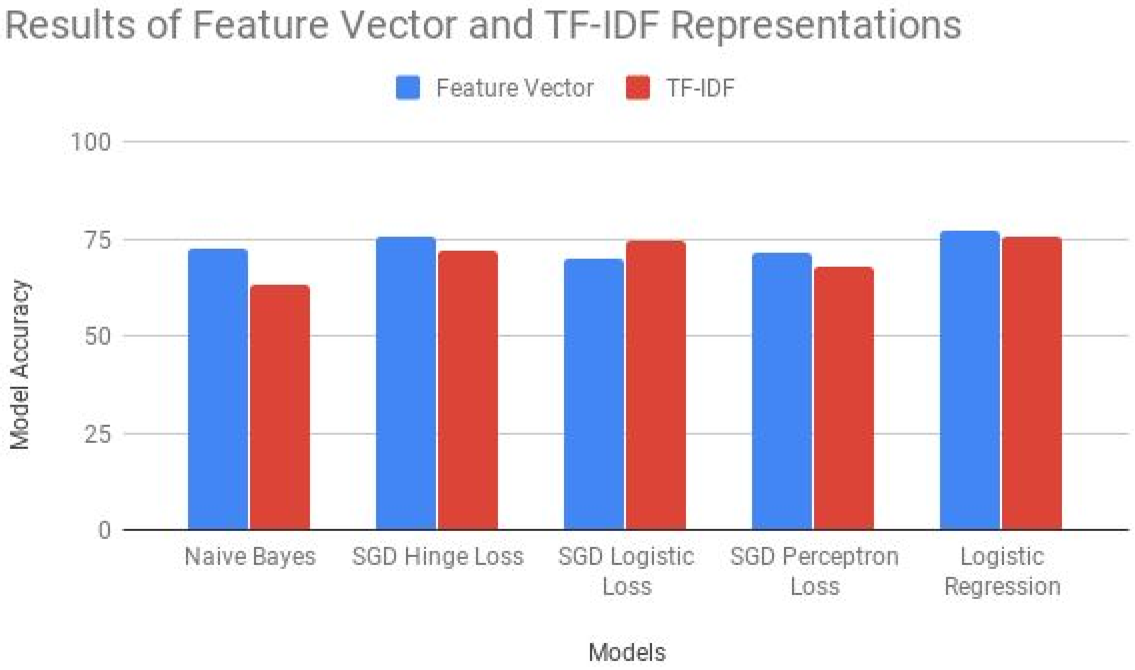
As seen, logistic regression that uses the logistic loss function proved to be the most accurate solution to classifying cuisine types based on recipe.

Overall Results and Analysis

Our baseline model, Naive Bayes using feature vectors, achieved 72.37% accuracy, and in the end, our logistic regression model using feature vectors performed best, at 76.95% accuracy. It is not surprising that the other classifiers performed better than the baseline, since Naive Bayes had to assume independence among features that often are not independent. In actuality, recipe ingredients are highly dependent on one another. Many ingredient combinations commonly occur together -- e.g. carrot, celery, and onion -- because of their complementary flavors, while others rarely occur together, either because they do not go well together, or their flavors are redundant.

While the performance improvement of logistic regression was expected, we were surprised to find that, for all of our models except SGD with Logistic Loss, using TF-IDF for feature representation did not offer any improvement. We suspect that the effectiveness of TF-IDF was diminished by the fact that an ingredient can appear in a recipe only once, whereas for other text classification problems, a common, unimportant word like “the” can appear dozens of times in a single document.

The figure below compares the results of each model and data representation on our training set.



Next Steps

Moving forward, there are a few ways we could boost our classification to improve the accuracies. The first, text cleaning, involves reducing the noise from similar text features by removing punctuation, such as hyphens, and suffixes and grouping together similar features. For example, there are many similar ingredients such as “sliced tomatoes” and “diced tomatoes” that could be grouped together. Along similar lines, it would be interesting to introduce pre-trained word embeddings such as Word2Vec as features, to capture semantic similarities among ingredients.

Another way to improve the classifications would be the ensemble the models by stacking and blending their outputs to improve the results.