What is Cooking?

Authors: Alexander Bobowski & Tyler Paulsen

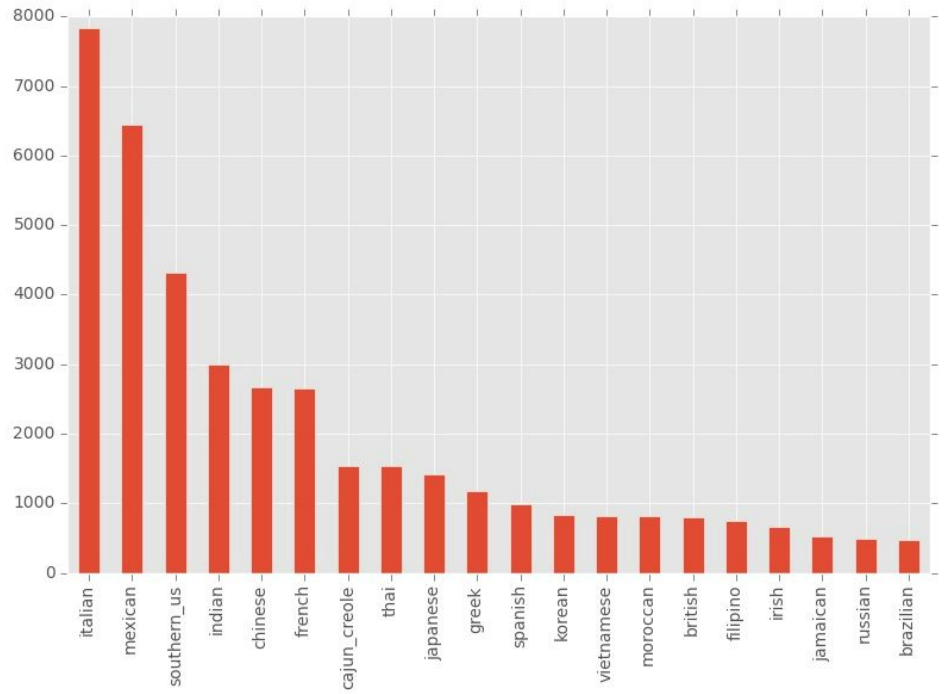
1. Problem Understanding

The first phase for the project was developing an understanding of the problem. Classifying cuisine based on the ingredients was unlike any of the classification problems we confronted in the homework assignments, so we needed to take a new approach. Many of the solutions used by other groups used a bag of ingredients approach, and this appeared to be a simple and effective algorithm for classifying recipes.

2. Data Understanding

Once the problem was understood, it was time to look at the data. Since the project was a challenge provided by Kaggle.com, there was plenty of information about what had been tried before. Many contestants had difficulty dealing with strings, which provided a good place to start working. The most helpful posts on Kaggle were the different graphs of how ingredients were distributed (Fig. 1.)

Figure 1:



The forums for this project helped direct our efforts towards areas of difficulty. There were many difficulties to overcome in order to implement a workable model, but contestants were friendly enough to point peers in the right direction.

3. Data Preparation

Preparing the data took the longest in comparison to the other steps. Knowing that the data was compromised of strings, and they can be inherently hard to deal with when trying to create a classifier, we knew we didn't want to get stuck trying to get the data perfect. When reading the forums for the project, there were people who have been working on the project for weeks, and haven't even begun to stop cleaning the data, so we tried to take a different approach.

Our approach was to find a library that could do some type of fuzzy word matching, so we did not have to spend much time cleaning the data, and let the library do it. The library we used for Python was FuzzyWuzzy (https://github.com/seatgeek/fuzzywuzzy). The python library had many useful functions relevant to the problems we faced; the most important one performed string comparisons. Given one string, and a list of strings, it found the strings from the list closest to the given string. Having the library able to pick the best match from a list of strings was immensely useful for creating a bag of ingredients.

The basic algorithm used to create our bag of ingredients was:

- for each data point:
    - for each *ingredient* in data point
        - get the cuisine
        - see if *ingredient* is like any other ingredient in current bag (we used 90%)
            - add 1 to hashtable for that ingredient. Counts how many times the ingredient was used.
        - else:
            - add new ingredient to hashtable

This algorithm had one obvious flaw that might be fixed in the next iteration: the first ingredient added if the key for the hashtable, even though it may not be the best key to represent the ingredients. The algorithm also made it extremely easy to apply to all the cuisines rather than just classifying one like our original intentions.

The first time this algorithm was run, it took around 3.5 hours to complete. Because of the issue stated above, it is our intention to run the algorithm again after utilizing a function in the fuzzywuzzy library that chooses a string that best represents a list of strings. When we run this

again we can use the representative words from the first classification run, which should provide more accurate string matching.

The program's resulting list of top ingredients for each cuisine provided us with our "bag of ingredients". The bag of ingredients was limited to 500 entries for each cuisine, but this is far too large in practice. With a bag of ingredients this large, a 100% match could be picked, but with the weighted score we use in the end classification, it would skew how accurate the ingredients represent the cuisine.

Each bag of ingredients was then run through a similar algorithm to reduce the number of ingredients. Each bag was iterated through, and the fuzzy string matching was performed again. If ingredients were 90% similar, it combined them into the the smaller string between the two. We choose to do the smaller string because it would keep the ingredients more generic, and the library loves short strings -- we learned this later.

The last step the the data cleaning process was to reduce the number of ingredients once again. This time, the number of ingredients was reduced based on how well the classifier performed. It determined that the magic number of ingredients per bag was 30, which provided the best classification available.

4. Data Modeling

The classifier for the project took the least amount of time. To score each cuisine, we took the percentage*100*(% ingredients the chosen ingredient makes up in the bag). A simple algorithm was used to classify the ingredient.

- Given an unclassified cuisine
    - for each ingredient in the unclassified cuisine
        - for each cuisine in the classifier
            - match ingredient to best ingredient in the bag
            - if < threshold skip
            - find the best choice (note 1)
        - add the weighted score hashtable<cuisine,score> for the best choice
    - get the best score from the hashtable, the key is our classification.

note 1: The fuzzywuzzy library can return choices that have words that are the same 'likeness'. To break this tie, we took the word that had the most weight associated with it.

The first time this was ran, the correct percentage was very low (19%). At this point, I realized why the professor told us not to try to classify them all. After a break of about 2 days, we relooked at how the fuzzywuzzy library did its extraction of the best word, and we found three things:

1. The matcher prefered exact matches in large strings.
    a. ex: "Meats",["Meats soup", "Meatloaf"]. "Meats soup" would classify much higher.

2. The default scorer -- the algorithm used to find best word -- uses three separate algorithms, and chooses the best score. We did not like this, because it seemed like the best score was not really a good indicator it was the best match.
3. The matcher is REALLY slow.

To solve this, we have to take a look into the library itself. With the default matcher parameters, it selects what scorer uses to score the words. The default scorer used three different algorithms -- this is why it is slow. Each algorithm had its own niche, and we did not like how it just picked the best score out of the three. We decided to used a fast simple scorer that just used the Levenshtein distance. Using a simpler scorer increased our correct percentage on test data to ~36%, still not good, but a huge increase provided by a simple change.

5. Evaluation

For the final evaluation, we used the Kaggle submission. We ran our classifying algorithm on the test data and got a value of 0.31285. For the first iteration of the classifier, this was not too disappointing.

6) Next iteration.

Now that we have a good idea on how the fuzzywuzzy library works, our next iteration should turn out to be much better. We can utilize it to create a better bag of ingredients; for example, making it pick the best ingredient that represents a list of similar ingredients.

While training the classifier, we noticed that certain cuisines always had a bad correct percentage -- brazilian, Chinese -- and we think if we add an additional weight to the ingredients that indicates how well it contributes to getting the classification correct, we could increase our correct classification percentage. Adding this weight would be similar to using a misclassification rate to pick what features are the best for classification.

any more ideas?