Assignment 4 - kNN Recipe Classifier
COMP 597

Charlie Summerscales

April 16 2015

1 Description of Approach

After several different models, I settled on a set of parameters that seemed to consistently give the best results. For feature vectorization, I chose to use feature hashing[1] along with bag-of-words to store term frequency. I tried using a traditional dictionary approach for storage, but by utilizing the hashing trick, I was able to increase other parameters such as the amount of neighbors that could be factored in to the decision. The basic gist of the hashing trick is that as long as an effective hashing algorithm is selected, then not only will collisions be minimized, but according to Zipf's law[2], rare words that may have a collision will have a minimal effect at best on the classification. Indeed, I tested feature hashing against dictionary storage and found no discernible difference in accuracy, but my test times did double when using the latter method.

I also settled on term frequency versus sets of words for my bag-of-words model, when I actually saw a slight dip in my accuracy. I believe this is related to frequency playing an important role in certain recipes. Consider an indian recipe which may have five different spices, all of which are preceded by the word 'grind'. Now, suppose there is a recipe in the training data that also has five ground spices, but the spice names form a disjoint set. The common factor would be the number of times that 'grind' appears, and so we become interested in this as a potential match.

As for my distance measure, I settled on a Cosine similarity approach after testing with both Euclidean and Jaccard distance metrics. This was more of a matter of trial and error rather than any notion of intuition.

In order to try and "train" the model towards more consistent results, I ended up running the training set against several different metrics. First, I ran the entire set through a term frequency program to determine the most common words amongst all recipes. My thought was to filter these from the training data so only the more meaningful words were left. Though some did not have any effect, and others had detrimental effects, there was a few words selected that helped to increase the accuracy of the model. Next, I parsed the data into seven different files, and ran each file through the term frequency program. This gave me the most common words amongst 'classes' of recipes. With these, I selected one word from each set that was both common in its own class, but rare in all others. These seven words had additional weight attached to them during the feature vectorization step in order to 'pull' any results that might have had those words, closer to potential matches.

[1] http://www.machinelearning.org/archive/icml2009/papers/407.pdf
[2] http://mathworld.wolfram.com/ZipfsLaw.html

Finally, I utilized a weighted voting scheme[3] to produce the predicted values. Other parameters that were tuned during the cross-validation phase included the number of neighbors utilized, and the size of my hashing table for feature hashing.

2 Confusion Matrix and Accuracy Estimates

After the final tweaks were made to the model, the last cross-validation test was executed using a leave-one-out technique. Model accuracy was 91.14%.

		Predicted							
		French	Italian	Indian	Chinese	Thai	Greek	Mexican	Total
Actual	French	2831	220	36	26	31	41	64	3249
	Italian	57	3237	12	18	22	43	23	3412
	Indian	42	47	3054	32	73	14	42	3304
	Chinese	33	32	33	3139	92	12	24	3365
	Thai	22	49	55	192	2892	17	28	3255
	Greek	55	99	22	19	19	1362	16	1592
	Mexican	81	50	42	22	56	5	3234	3490
	Total	3121	3734	3254	3448	3185	1494	3431	

Figure 1: Confusion Matrix for final Cross-Validation results

[3] http://ojs.academypublisher.com/index.php/jcp/article/view/0605833840