Project 2: Applying Multinomial Naïve Bayes to Recipe Data due April 8, 2018
CSC 420 – Machine Learning

Jordan Turley

Overview

For this project, we applied the method of Multinomial Naïve Bayes for classification to the Wikia Recipe database. We wanted to be able to classify a content page as a recipe or not. We filtered out any non-content pages, then hand-classified 100 content pages as a recipe or not. After applying Multinomial Naïve Bayes, we were able to achieve an F1-score of 0.66. While this isn't bad, it could have been better. I discuss our results and how this score could have been improved in later sections.

Data Preparation

The source of our data was the Wikia Recipe database, which we downloaded from the Wikia Recipe website. The file is in XML, and contains several thousand pages, some content and some not, from the website. I created a helper program called 'recipe_classify.py' to filter out non-content pages and allow the user to hand classify a certain number of content pages as a recipe or not.

The first step we took was to filter out all non-content pages. To do this, we looked at the title of the page. If we split the title on a colon (:) and look at the first part, we can tell if the page is a content page or not. There were a few recipes that contained colons, but most of the time, the title looked like "User blog", "Recipes Wiki talk", "Category talk", etc. and we could discard these. Then, the user was able to hand classify 100 content pages as recipe or non-recipe. These results were stored in a file named data.txt for use by our classifier program.

Method

We applied Multinomial Naïve Bayes to our hand classified data. The Naïve Bayes model uses a 'bag of words' to count the frequency of every word in the document, ignoring the sequence of the words. Then, the model tries to determine which 'bag' the document is more likely to come from: the recipe bag or the non-recipe bag.

To begin, we read the classified data from the file data.txt generated by 'recipe_classify.py', randomly shuffled the data, and split the data into a training and test set. We trained the Multinomial Naïve Bayes model on our training data, and then used our test set to evaluate the performance of our model. Our results are given in the next section.

Instead of implementing Multinomial Naïve Bayes on our own, we used the scikit-learn toolkit in Python. We used a Pipeline object to make our code more concise, which used CountVectorizer and TfidfTransformer objects to convert our documents to bags of words. We then used a MultinomialNB object as the model for classification.

Results

To evaluate our model, we calculated the error rate, precision, recall, F1-score, and support, as well as generating a confusion matrix for analysis. We can see the results in the following tables.

Error rate = 6/20 = 0.3

Table 1: Evaluation Metrics

	Precision	Recall	F1-score	Support
Y	1.00	0.33	0.50	9
N	0.65	1.00	0.79	11
Avg/total	0.81	0.70	0.66	20

Table 2: Confusion Matrix

	Predicted Class: N	Predicted Class: Y
Actual Class: N	3	6
Actual Class: Y	0	11

Analysis:

We see an error rate of 0.3, which is higher than we would like, but is not terrible. However, the error rate does not mean as much as we would like it to; since there are more recipes than non-recipes, simply saying everything is a recipe could result in a good error rate. We can also look at the results in Table 1. We see that our results are not bad, but there is room for improvement. We will discuss this later.

If we look at our confusion matrix, we see that there were three instances where the class was predicted to be N and it was correct, and eleven where the class was predicted to be Y and it was correct. There were no false negatives, i.e. if the classifier was given a recipe, it was always right. This is good. However, we did have a handful of false positives. We can look at examples of correctly and incorrectly classified results. First, we will look at some correctly classified instances:

Y: Description festivity cakes Prepared for religious celebrations Ingredients 500g flour…
Y: Description Sour cream pancakes Sounds odd but these are really good Not just for…
N: '''Steaming''' is cooking by steam Steaming is a preferred cooking method of health conscious

We can see that our classifier does a good job classifying actual recipes. We also see some definitions that are correctly classified as non-recipes, such as a technique of cooking. These show that our classifier is doing what it should be doing in most cases. We can also look at our incorrectly classified instances:

Predicted Y: REDIRECT ham bone
Predicted Y: REDIRECT oka I'a Raw Fish in Coconut Cream
Predicted Y: A tomato based Mexican sauce made up of tomatoes chile peppers and onions It is a sauce…

This gives us more insight into the errors our model is making. We can see that redirect pages can cause the model to predict a Y when we can clearly see that these are not recipes. Because they contain the name of a recipe and words that are used in recipes, this can cause the model to see them as a recipe. Also, we see that some food definitions, like this definition of picante, can throw the classifier off, which is understandable. Many of the words that are used are the same words that would be used in a definition, like tomatoes, chile peppers, onions, etc. It is easy for a human to see that it is not a recipe, but it is a lot harder for a computer to see.

Shortcomings, Improvements, and Future Work:

There are several things that could be done to improve this model in the future.

1. More data would have made our classifier more accurate, without a doubt. If we had 1000 or 10000 data points instead of only 100, we would have seen much better performance out of our model. However, this is very tedious, as it must be done by hand.
2. We could try different text formatting schemes. In this experiment, we removed all non-alphanumeric characters and replaced certain punctuation with a space, such as dashes or [[and]] characters indicating a hyperlink. We could experiment by leaving some or all the punctuation and see if this helps the classifier perform better.
3. We could remove all of the 'REDIRECT' pages; they seem to be rather useless considering they contain no content, but cause the classifier trouble. The classifier may be able to classify these as non-recipes with more data, but this would need to be something we look at.
4. We could compare our Multinomial Naïve Bayes model to another model, such as a Support Vector Machine. It is possible that a different type of model is simply more suited to this data and will almost always do a better job.

Conclusion:

Overall, I am happy with the results we achieved. While our results were not great, I did not expect them to be great with only 100 data points. It would be interesting to classify more data points or pool all of the data from the class to see how much this improves our model. It would also be interesting to try different text formatting schemes, or compare the Multinomial Naïve Bayes model to another model. I feel like I learned a lot from this project. Project 1 helped me to see how the scikit-learn toolkit works, but this project allowed us to apply the model to actual data and go through the process of preparing the data and feeding it to the toolkit, then see our model does fairly well at predicting.