Ankit Gupta, Hirsh Jain, Nishant Kakar, Emily Wang
Final Project Functionality Checkpoint – Writeup

Progress

As of now, we have completed the core features of our algorithm. The Naive Bayes algorithm is fully implemented, and despite our limited training set, the algorithm successfully identifies liberal and conservative media sources, provided that the source material is drawn from very partisan news. No significant bugs have been identified, but accuracy and precision could be improved with larger training sets.

We implemented our algorithm in many steps. First, we wrote a web parser which, given a url, returns a dictionary of the word counts in the relevant parts of that page. We then iterate through a list of urls, and built a master dictionary, who keys are each of the words in all of our files. This gives us an overall word count. Then, we create a matrix whose dimensions are the number of articles by the number of words, and iterate through the set of urls again, converting each dictionary into one row of this matrix (many of the values may be zero). Then, we iterate through the matrix quickly, and create vectors (arrays) representing the probabilities of each word given a class variable (republican or democrat). Lastly, we can actually classify a new article by determining its word counts, taking the dot product of its word vector with the vectors we calculated before, and determining which class has the higher dot product. In order to implement this, we had several different files to organize our code. One file has all of the parsing information. Another file builds the matrix. Another file creates the vectors. One more file implements processing a new url and classifying it as liberal or conservative based on the trained data. Lastly, the file “naive_bayes.py” is the one we run the program on, and it properly accesses all of the relevant files in the correct order. We also have files which have our training data sets of urls with conservative and liberal files, along with a list of urls that we will classify by our algorithm. Lastly, we have a text file with the most commonly used words in English, that we will factor out in the classification algorithm so that they do not interfere with the more important words.

Problems/Issues

We haven’t run into any significant bugs or errors, but we will definitely reach out to Angela, our TF, if we do.

Otherwise, we have encountered some smaller bugs, but we have an idea of what might be causing them.

One problem we have encountered is an issue within parsing. Two team members have an older version of Python (2.7.2), while two members have an updated version (2.7.5). For some reason, some websites can only be parsed successfully (using Beautiful Soup) in the updated version. For example, we were trying to parse liberal articles from ThinkProgress, but the code ran successfully only on Ankit and Nishant’s computers.

We also ran into a slight issue regarding common words. Initially, very common (and nonpartisan) words (such as “and,” “or,” “but,” “the,” etc.) were causing the distinctions between liberal and conservative dictionaries to be less pronounced, and our program was not as accurate at predicting the partisan bias of extremely partisan news sources such as blog posts from the Heritage Foundation. However, we implemented a quick matrix cleaner that eliminated

all of the most common words from total word counts, and this seems to have addressed the issue pretty effectively.

Teamwork

We have been pretty successful in divvying up the workload. We learned the algorithm together and developed the pseudocode as a team, so we're all familiar with the concepts in the project. Since our schedules have aligned well this week, we were able to spend several concentrated hours working together on developing concepts and algorithms as a team. While some modules were coded by individual team members, we all aided each other in fixing bugs and understanding how the modules worked with each other.

However, we have modularized our code, so if our schedules are less compatible next week, we will be able to delegate certain tasks to individual team members.

Plan

Since the core of our final project is implemented, we'll mostly be working on extensions. We aim to complete these extensions by Friday; however, the most urgent extension involves improving our training set, and this should be completed by Tuesday.

Since we still need to understand the concepts behind stemming, web crawling, and cross verification, we have not yet distributed the work amongst team members. However, we will adopt an approach similar to this past week's, where we learned concepts as a whole group and implemented smaller parts of code individually. Continuing to modularize our code will simplify this process.

- We need to increase the size of our training set. All members on the team will be involved in this. This will mean we need to get a large number of liberal documents and a large number of conservative documents (we are still trying to determine how many we need exactly), along with a number of documents to run the classifier on.
- We want to implement stemming, which would take similar words ("like", "liked", liking") and consider them as one word, rather than processing them separately. This should increase the accuracy of our code.
- We want to implement a web crawler. This would take all articles from websites we have identified as conservative or liberal (for example, Think Progress or The Heritage Foundation, which are the two sources we're currently using), and add the URL's to the collection of conservative and liberal URLs.
- Lastly, if possible we will implement cross-verification (running the classifier by training it with a subset of the training set and running it on the rest).