Brandon Mueller, Jeff Ross, Donovan Rennaker

CS-380

AI-Final Report

12.01.2016

This is our Artificial Intelligence Final Report for CS-380. Our task is to choose a publically available dataset and to create a Python program that completes the preprocessing that is necessary to run said data through at least one of the sklearn algorithms from machine learning. The machine learning algorithms that we were offered to choose from are as follows: Naïve Bayes, kNN ( k-Next Neighbors ), SVM ( Support Vector Machine ), Decision Trees, or Convolution Neural Networks.

**W**hich algorithm did we use, why did we choose that one, how does it work, and how does it compare to the other available options?

The algorithm that we choose to use as a group is Naïve bayes, and the reason we choose this algorithm is a long story. Initially, we chose to use Naïve Bayes in order to try to predict the number of likes that a forum post should receive based on training posts that came from a political forum on Reddit. We have just under ten datasets, where the smallest of the datasets contains between five-hundred and one-thousand messages and where the largest of the datasets contains more than fifteen-thousand messages. After trimming down the datasets, we were left with the titles of the two remaining columns which were: “likes”, and “message”. After writing our program to preprocess that information, our classifier was printing out some very convincing information that one would consider to be reasonable numbers of likes that you would see on a Social Media forum. After checking in with Professor Licato about our results, we were informed that although our information was convincing, it was not correct. The reason the information was incorrect is because Naïve Bayes serves one major purpose: to classify information based on categories, so our program was not actually predicting likes that a comment should receive but it was instead matching a test message as closely as possible with a message that was part of the trained dataset. To determine the numbers of likes that a comment on Social Media should receive, the Professor suggested that we use the skLearn Regression Algorithm. After spending a while re-writing sections of the program so that our dataset would run using the new algorithm, we ran into another problem: The scores that were being returned were nowhere close to the scores that the original Reddit post had received. After tweaking the parameters for the algorithm and finding no success, we asked Professor Licato once again for assistance. After further explaining the problem, mentioning the things we have tried, and results that those attempts were producing, the Professor indicated that the most likely reason why we are not getting accurate results is because the algorithms work assuming that the datasets are normalized. Since our datasets, all of our datasets regardless of size, are based on forums responses, we found it to be very unlikely that our datasets are normalized. Professor Licato suggested giving Naïve Bayes another try, but instead of predicting an exact number of likes that a post should receive, we should create a small number of categories such as “low-likes”, “medium-likes”, and “high-likes”. After being given this recommendation, we changed our program so a post that receives a small numbers of likes would return 0, and a post that is moderately popular would return 1, and the largely popular posts would return 2. After testing this, it appears to gives us some reasonably accurate responses. However, because it’s Naïve Bayes, it appears to gives us the results that we are looking for about half the time, which is consistent with our SVMs assignment where Naïve Bayes had an accuracy rate of about 55%, while SVMs was 85% accurate.

How does the Naïve Bayes Algorithm Work? Naïve Bayes is trained by separating past information into categories, and then matches new information based on that information. This algorithm, compared to the others, appears to be the least accurate of the ones that we were allowed to choose from, but after running our datasets through Regression and kNN, we were not receiving the correct scores. So for normalized datasets, all of the other algorithms are far more accurate, but due to the nature of the data from social media, such data contains terrible formatting errors, misspellings, and special characters such as emojis; which are difficult to completely pre-process.

**W**hat mistakes did we make, how did we go about fixing those mistakes, and what were some of the unexpected difficulties?

At this point in time during our project, a vast majority of the mistakes that we made was in the writing of the program that would be used to process our datasets and provide results. The other mistakes made occurred while formatting the datasets for both the Regression algorithm and finally for the Naïve Bayes algorithm.

One of the first issues or mistakes that we encountered while working on this project was that we forgot that we were working with posts from a social media site, and as such, just about every post contains characters that were unreadable by Unicode and / or uft-8 formats. The reason for this was because users on these sites were posting more than just messages, but also pictures, emojis, videos, and other contents that our regular text processing program does not understand. The information as mentioned above was crashing our program. So since those pieces of data have no value to us, we ignored it and moved on. The second issue that we ran into was the fact that the datasets that we were working with was pulled from the Reddit website using an automated script of some sort and as such, it grabbed all of the escape characters along with the text. In order to get by this hurdle, we had to make use of the “re” library, which is similar to the split function in Python, only that “re” allows us to chop off more unwanted characters all at the same time using regular expressions ( regex ).

Just to take a breather here between sections, we easily ran into five or six pretty substantial issues that slowed us down.

The third issue that we ran into while working on this project was fitting the data. This was an issue for several reasons: first, because the arrays were not sized correctly; two index out of bounds; and three, reading extra data that we did not intend to use. For the first of the three reasons that the data would not fit together was because we either didn’t read far enough into a file or we tried to read a little too far into a file. The second reason of the three is that we popped some information that we determined was not needed and once again threw off the sizes of the arrays. The third reason was actually the biggest pain to work with because, as I mentioned for the first question, each dataset has titles for the columns, which were: “likes”, and “message”. The only information that was important to us was the number of likes a message received, and the actual message itself, not the column headers. Our solution to that was to remove that text from the datasets.

The next issue, which would put us at our fourth issue, was reading too much data. Instead of reading the lines from the file and separating the labels from the actual text, the vectors received all of that information, which threw off our program. Our solution to that problem was to pop the first index of a line from the file. This is because we knew that when the file was read, it was returning an array with two indexes ( csv is comma delimited ), and we also knew that the first piece of data on every line was a number. So, instead of running a separate function to create the labels array, we did this while the dictionary was being populated.

Our fifth issue, we noticed that our dictionary had a certain word that was being found hundreds of times in the text for a single message. We knew that couldn’t be right. So, after looking into it, we determined that our program that counting a space as a word and was counting it. This very easily could have been throwing off our results, so spaces are not counted as a dictionary term.

Finally, the sixth and final major issue for our program was… memory. Since the dictionary was so large, even for the smallest of our datasets, our program would claim a ton of memory to run through. This posed a major problem, because even my computer at home was unable to run the largest dataset even with 16GB of RAM at my disposal. At this point, we asked Professor Licato for recommendations. We were told to trim down the dictionary size. We were able to do this rather quickly, because in doing so, we were not interfering with the vector creation or label creation. In order to trim down the size of the dictionary, we had to add one additional function which would keep track of how many times each word appeared in the initiate dictionary. Once we had that information, we re-populated the dictionary with the words which appeared in our dataset ten or more times. Since a very large number of words only appeared a single time, the dictionary was reduced by 90%, and in doing so, many of the horribly misspelled words were eliminated.

As was mentioned earlier, the last of the problems that we encountered while working on this project, was making a converted copy of our datasets so it could work with Naïve Bayes. In order to do that, I had to find a Visual Basic script that could be used as an Excel Macro, which would change all numbers within a certain range to another target number. The find and replace option in Excel did not work the way I needed it to.

**W**hat parameters did we use and why, and what algorithms or functions did we have to write in order to achieve the goal of our project?

Our program that we wrote to process our datasets is not incredibly large. It is actually less than 150 lines of Python code. Other than the file we loaded in to retrieve the data from our dataset, our program uses seven parameters: A dictionary that keeps track of the words and indexes, another dictionary that keeps track of the number of times each word appears in the dataset which assists in trimming the dictionary later, an array that holds a single line in the file, a double array that holds the vectors, and one more array which holds the labels. The last two parameters are simply switches. The first switch controls whether files are written out, which tells us what type of data we are running through our program after pre-processing the dataset. The seventh and final switch is just a flag that prevents the wordUse Dictionary from being populated more than once.

The reason we chose to use the parameters that were mentioned above is to hold the data that we extracted and processed, and to control how many times certain data was processed. When I say how many times the data was processed, I am referring to the wordUse dictionary This is because the wordUse dictionary is part of the “find\_word\_index” function and is referenced in two different areas of our program. To keep the wordUse dictionary from counting all words twice, this flag was added to control this.

Our program consists of eight algorithms or functions that we used to find an answer to the question that was had on our datasets. The first function is the “save\_words” function, and its primary purpose is to generate the dictionary of all words that appeared in our dataset. However, the function did more than just load the file and store the words, the text that came in from our file wasn’t exactly ready to use immediately, so we had to sanitize the data and keep track of the labels and the number of times each word that appeared in the file. We kept track of the number of times words appeared in the file because we would need that information for a later function. The second function that we used was the “find\_word\_index” function, and had a very simple purpose to take in a string (namely a word) as a parameter, check if that word is in the dictionary, and then return what index in the dictionary that the word was at, or return -1 to indicate that the word was not found. This function proved to be very useful when generating the very large vector which contained numerical representations of the comments that were on each line of our dataset file. This function was also used when the user enters a custom comment into the console / terminal for a category prediction. The third function used is the “save\_vectors” function, which has three main purposes: one, to trim the dictionary using the wordUse dictionary; second, to populate the vectors; and third, to write the vectors out to a file if the write field is set to “yes”. The fourth function used is the “save\_Word\_Use” function, which only has one purpose, which is to write the wordUse dictionary out to a csv file so it contains two columns: the word and the number of times that word showed up in the whole dataset. Before I continue to the next function, I want to make sure that our readers are informed as to why the program can be set to write out all of this information. There are a total of four files that are written: words.csv, vectors.csv, sentiments.csv, and worduse.csv. Once these files are written out, they are not read back in again, this information is written out purely for debugging purposes so we can see what information the program is working with and to detect any anomalies in said data and consequently fix the bugs that causes that data to be generated.

**W**hat are the strengths and limitations of our final approach?

…

**N**ow that we know better after having written this program, tested it extensively, and correctly many bugs and difficulties, what would we do differently to get say: get better results and / or to make the program run faster?

…