Carl Mcanich
Justin Kachornvanich
Tyrone Williamson
Dr. Kurdi
CS373
12/2/22

Project Spam

Spam email is a problem that everyone from businesses, government officials, to regular people have to be careful about everyday. Bad actors try to infiltrate into government networks to steal secrets, scammers pretend to be rich relatives or prince's to try to swindle money from the unsuspecting, or hackers can overload systems to cause massive problems. Spam detection is essential in stopping a lot of these problems and early techniques like Blacklisting and Signature based System Spam detection were developed but proved to be insufficient to tackle the problem. Later text based techniques were introduced in order to get efficient, consistent and accurate results by filtering through the words of an email and running these words through a dataset of example words.This helps sift through the various emails and decide if it is spam or not. We approach this project by taking the spam.csv file and splitting the dataset into spam and ham text. Next we tokenize and remove all of the stop words from the given text and stem the remaining words.

We Split the data into spam and ham and made a list of lists for each one where each index of the list is another list that holds a single email as a string. Next we tokenized the spam and ham text, removed the stop words from spam and ham, stemmed the remaining words from both, and got the number of times each stem occurs in the entire dataset. Then we made two dictionaries, one for spam and one for ham where the keys are the stems and the values are the number of occurrences of the stem in the entire dataset. We then sorted the counts in descending order and cut the dictionaries to get rid of everything but the first 50 stems (the top 50). We went on to merged the two dictionaries to get the top 100 stems, created a blank dictionary where the stems are the keys and the values are empty arrays, and then checked each stem for its occurrence in the cleaned emails (tokenized, removed stop words, and stemmed) and stored each stems occurrence counts in a list per email

We read in a csv file that consisted of the top 100 unique stems and split the x data into columns and the y into class. When we scaled the data to have mean 0 and variance 1, which is important for convergence of the neural network. Next we Split the data set into training and testing. constructor of the class takes the input size as a parameter, we set the activation function, softmax turns a vector of K real values into a vector of K real values that sum to 1 train_feddForward() uses the Adam optimization algorithm, which is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing. It is used to update network weights based on training data. We also use the CrossEntropyLoss() function which is useful when training a classification problem with C classes. If provided, the optional argument weight should be a 1D Tensor assigning weight to each of the classes. This is particularly useful when you have an unbalanced training set. We need to set the gradients to zero before starting to do back propagation because PyTorch accumulates the gradients on subsequent backward passes. We use a Context-manager that disabled gradient calculation. Disabling gradient calculation is useful for inference, when you are sure that you will not call Tensor.backward(). It will reduce memory consumption for computations that would otherwise have requires_grad=True. We use plot_result() function which plots both the accuracy and the loss given the number of epochs

Both Decision tree and Random Forest methods returned similar results. DT algorithm data and the results we got were as follows: train data accuracy = 0.9584615384615385, and test data accuracy = 0.9497607655502392. RF algorithm our data and the results we got were as follows: train data accuracy = 0.9584615384615385, and test data accuracy = 0.9509569377990431 The Neural Networks both gave us two different outputs every time we ran them so we did not get a conclusive result. We suspect this is because it is a neural network but are not sure

For further work, even though the results were very close to one, we could have used more data to hopefully improve those results a bit more. We could have also used more machine learning techniques such as Long-Short Term Memory for possibly the best rests due to its recurrent property. Along with further work, we learned a lot through this project. First we learned many new shortcuts that we could use pandas and numpy for. Lots of built in functions did jobs that we initially did with many lines of code. Secondly, this project reinforced the topics of DT, RF and NN for us. Understanding what pieces to change in the code and what data each technique needed aided in understanding them much more. Lastly, we learned natural language processing techniques such as tokenization, stemming and removing the stop words.