Hans Tananda 946182011

This project aims to classify which domain a research paper belongs to (Archaea, Bacteria, Eukaryota, or Virus) given abstracts as the feature. The Multinomial Naive Bayes classifier is used as the classifier, where the words frequency will be used as the features of the classifier. Therefore, the data need to be preprocessed to match the classifier input requirements. This is done through a series of steps. First of all, the abstract sentences are split according to spaces. Then, those words are transformed into lowercase and all but alphabet characters are removed. This is done so that words like “DNA,” , “d-n-a”, and “dna.” will be mapped to the same word “dna”.

After that, the overall number of word count in the whole dataset is calculated. This is done so that the total number of unique words can be known as well. By doing so, those results can be stored in a numpy array instead of a dictionary. This can lead to better performance and lower memory usage for our implementation. The overall number of word count per class is then calculated. By doing this, we can start training our classifier by calculating the probability of a word appearing in a certain class. Thus, our training has been concluded. The training for the whole dataset took around 5 seconds to complete.

The test data follows the same preprocessing method as the training data, and then the probability of each test instance in a certain class is calculated. Then for each instance, the highest probable class is picked as the predicted class for that instance. When 10-fold cross-validation was executed, an overall accuracy of 82.4% was obtained.

After successfully implementing the classifier, some further enhancements are introduced and tested. First, we tried to transform the frequency based on length, since the rationale behind this is the fact that the longer a document is, the more likely it is to contain certain words multiple times. However, after we tested this implementation, the accuracy dropped quite badly instead (76.77% accuracy after crossval test performed). Therefore, we decided not to use this method.

The next step that I decided to do was to try ignoring stop words from the abstract. Therefore, a list of stopwords from a github gist is used to help identify which words are the stopwords. Then, a logic is added during the preprocessing to ignore those words during the calculation. Apparently, we successfully achieved an overall accuracy of 87.3% with 10-fold cross-validation. There was no significant difference in speed of the training and prediction after this method was added.

After knowing that ignoring stop words can be useful, I would like to try to use the generalization of this idea. Therefore, the next extension to be tested was to downweight common words. This could be useful since there may be some common words that are unlikely to be related to the class but may appear due to random noise. In our case, it may be words like “biology” and “anatomy”, both of which can appear in all 4 of our classes. Fortunately, this method increases our overall accuracy by around 1.2%. However, now our training time took around 90 seconds per fold. I still decided to use this one to be the final one to be used as our classifier.

On a side note, NumPy library, especially Numpy Arrays, was used for most parts in this project where possible since its data structures are much faster than traditional Python data structures. I tried to use Numba to boost and parallelize this project, but failed to do so since I was using classes and it’s quite complex to transform those classes into classes that can be supported by the Numba JIT compiler. I initially used Pandas to load the data easily and fast. However, since there was a query in piazza stating that we should not use this library then I transformed it by making my own csv reader using an internal python csv reader. The experiments and testing can also be found under this github repository.