Lucas Lopilato

328886-7

Machine Problem 1 Report

Architecture

For machine problem 1, I wanted to abstract network training from probability calculation and testing. As a result I ended up with two objects NaiveBayesClassifier (NaiveBayesClassifier.py), Trainer (Trainer.py) and one module stemmer (stemmer.py). NaiveBayesClassifier.py acts as the main point of entry for the program, and instantiates the NaiveBayesClassifier object given the proper arguments are supplied.

When NaiveBayesClassifier is created, it creates and holds a reference to Trainer. As soon as Trainer is instantiated, the network is trained. Trainer holds an in memory representation of all words, split up by class (positive or negative). Trainer also holds all methods for preprocessing (see section). If the stem option is enabled on the NaiveBayes Classifier, the trainer also instantiates the stem object.

The Stemmer is a partial implementation of the Porter Stemming algorithm. If it is enabled, it works to simplify some words in to more common forms that may appear in the text. If the option is not enabled, the stemmer will not operate or be instantiated.

As soon as NaïveBayesClassifier has instantiated Trainer, it begins working on the testing data and evaluating results. NaiveBayesClassifier holds all functions necessary to guess the classification of new tests, and it runs tests on the training data as well as the testing data, outputting the expected result.

Name	Options	Function
NaiveBayesClassifier	**pbayes** – use classical bayes method **pmultibayes** - use polynomial bayes method **pmissing –** use counting method for missing words **top10 –** display the top 10 decisive words rather than test	• Instantiates Trainer • Guesses output of tests • Runs timetests and reports results
Trainer	**stop –** filter from list of stop words **stem –** use stemming when reading in words	• Instantiates Stemmer if option enabled • Cleans and filters inputs with preprocessing and/or stemming and stop

		words
Stemmer		• Takes word and returns the stemmed version

Preprocessing

For preprocessing, I implemented three features: Prescreening, Stop Words, and Stemming. Prescreening takes an input string from the file, strips leading and trailing whitespace then lowercases all characters. Then it reads the entire string and removes all non-lowercase symbols and numbers excluding whitespace.

If stop words are enabled, the trainer builds a set of all stop words. If a word shows up in the input that exists in the stop word set, it will be ignored. Lookups on the set are O(1), so do not adversely affect the runtime of the algorithm.

If stemming is enabled, my version of the porter stemming algorithm is enabled. After stop words are screened, the word is taken and stemmed. If it fits the rules I have implemented, the stem rather than the word is added to the collection of either positive or negative words (depending on label)

After preprocessing, the each cleaned word is added to a respective class dictionary (positive or negative) based on the label at the end of the line.

Model Building

The Trainer builds the model. It reads in the input file, reads the label, and then applies preprocessing to the input string depending on options. From there the trainer builds and holds in memory a dictionary for the positive class and a dictionary for the negative class. If the training label is positive, all words frequencies are added to the positive dictionary or the negative dictionary if the label is negative. When building the dictionaries, the trainer also counts certain attributes that will be used in calculation to reduce runtime.

Results

Results depend on enabled options. My classifier has up to 3 methods to choose from: Bayes' Method, Multinomial Bayes Method (with a choice of smoothing), and a weighting of which sets do not contain certain words. Depending on options given, the classifier will run enabled methods, sum up the guesses, and then divide them by the total. If a majority believes a certain way, that way will be chosen. My best performance on the given set of information is .978 training and .946 testing. My best performance of the remote set of information is .835. For python, I also have a strong runtime of 3 seconds on the remote set.

I found my best words by taking the ten largest differences between both classes returned by the multinomial bayes method. In order of greatest to least: right, bromwell, adults, classic, ran, profession, sack, years, high, isn't.

Challenges

When testing my options, I found that many of my options improve performance on the local training set, and degraded performance on the remote training set. As such, I have optimized the submitted version to perform for the remote dataset rather than the local.

I also was challenged finding optimizations in my code to improve runtime as best as possible. I found that the more optimizations I needed, the more information I needed to gather and save from the initial training data, resulting in a large amount of memory and member variables of each object. Despite this setback, I optimized the code for runtime and pulled as many repeated calculations out into the training phase.

Weaknesses

Due to the number of options and optimizations, my system is not completely optimized for the remote testing set. I originally gathered a list of stop words from a remote source (cited in Trainer.py) and pruned a good amount of the words. From there, however, I believe there are more words I could remove from the stop words to optimize performance.

The stemming algorithm is also fairly rudimentary, and only implements a few basic rules of the Porter Stemming algorithm. With this, the stemming only has a limited range of words it may operate on rather than creating a best-case operation on the English vocabulary.

Testing

For testing I used the built in python unittest framework to test the workings of each of my classes to ensure that my assumptions were correct. I did have to wipe my test file for the main method after I added options due to unforeseen consequences, but the NaiveBayesClassifier class is thoroughly tested.