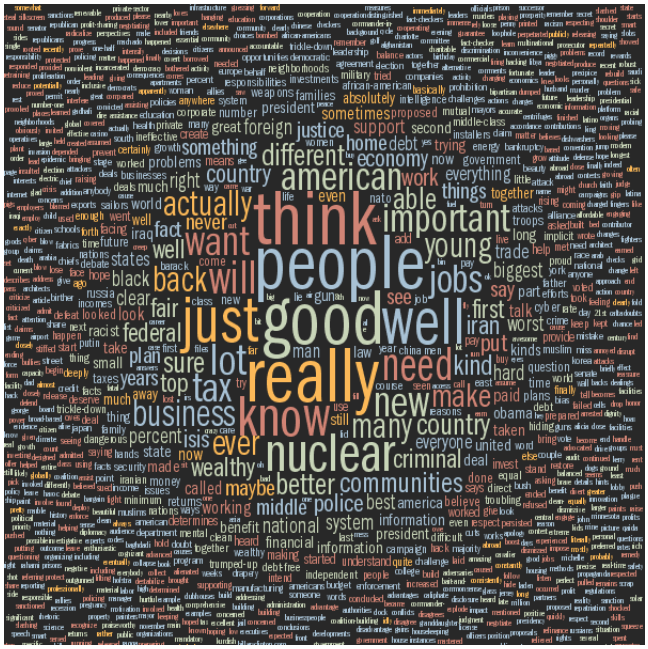

CS440/ECE448 Fall 2022 MP2: Bigram Naive Bayes

Due date: Monday September 18th, 11:59pm

Suppose that we're building an app that recommends movies. We've scraped a large set of reviews off the web, but (for obvious reasons) we would like to recommend only movies with positive reviews. In this assignment, you will use the Naive Bayes algorithm to train a binary sentiment classifier with a dataset of movie reviews. The task is to learn a bag of words bigram model that will classify a review as positive or negative based on the words it contains.

General guidelines

This assignment is written in **Python 3**. You should be using **Python version 3.8** for all assignments in this course, due to compatability with Gradescope's autograder. If you have never used Python before, a good place to start is the Python tutorial.

Your code may import modules that are part of the standard python library. You should also import nltk, numpy, and tqdm. You can install modules locally using the **pip3** tool.

We will load nltk so that we can use its tokenizer and the Porter Stemmer. You may not use any other utilities from the nltk package unless you get permission from the instructor. More information about nltk can be found at the NLTK web site.

The tqdm package provides nice progress displays for the main loops in the algorithm, because they can be somewhat slow when handling this amount of data.

For general instructions, see the main MP page and the syllabus.

Provided Code

We have provided a (zip package) containing all the code to get you started on your MP, as well as the training and development datasets.

Submit only your **bigram_naive_bayes.py** file on gradescope.

The main program mp2.py is provided to help you test your program. The autograder does not use it. So feel free to modify the default values of tunable parameters (near the end of the file). Do not modify reader.py.

To run the main program, type **python3 mp2.py** in your terminal. This should load the provided datasets and run the bigramBayes function on them. Sadly, it's not doing the required training and just returns the label -1 for all the reviews. Your job is to write real code for bigramBayes, returning a list of 0's (Negative) and 1's (Positive).

The main program mp2.py accepts values for a number of tunable parameters. To see the details, type **python3 mp2.py -h** in your terminal. Note that you can and should change the parameters as necessary to achieve good performance.

Submitting to Gradescope

Submit this assignment by uploading `bigram_naive_bayes.py` to Gradescope. You can upload other files with it, but only `bigram_naive_bayes.py` will be retained by the autograder. We strongly encourage you to submit to Gradescope early and often as you will be able to see your final score there.

Policies

You are expected to be familiar with the general policies on the course syllabus (e.g. academic integrity) and on the top-level MP page (e.g. code style). In particular, notice that this is an individual assignment.

Dataset

The dataset in your template package consists of 10000 positive and 3000 negative movie reviews. It is a subset of the Stanford Movie Review Dataset, which was originally introduced by this paper. We have split this data set for you into 5000 development examples and 8000 training examples. The autograder also has a hidden set of test examples, generally similar to the development dataset.

Bigram Mixture Model

You will implement the function bigramBayes that computes the mixture of unigram and bigram bag of words models. Each bigram b_i is a sequence of two consecutive words from a training or test review. Your bigram code should be very similar to your unigram code, except that you're looking at pairs of words rather than single words. So the probabilities for bigram and unigram models look like this:

$$P(\text{Type} = \text{Positive}|\text{Words}) = \frac{P(\text{Type} = \text{Positive})}{P(\text{Words})} \prod_{\text{All word pairs}} P(\text{Word Pair}|\text{Type} = \text{Positive})$$

$$P(\text{Type} = \text{Negative}|\text{Words}) = \frac{P(\text{Type} = \text{Negative})}{P(\text{Words})} \prod_{\text{All word pairs}} P(\text{Word Pair}|\text{Type} = \text{Negative})$$

Then you combine the bigram model and the unigram model into a mixture model defined with parameter λ:

$$(1-\lambda)\log\left[P(Y)\prod_{i=1}^{n}P(w_i|Y)\right]+\lambda\log\left[P(Y)\prod_{i=1}^{m}P(b_i|Y)\right]$$

The input to bigramBayes includes two Laplace smoothing parameters, one for the unigram model and one for the bigram model.

The parameter λ controls how much emphasis to give to the unigram model and how much to the bigram model. Choose the value of λ that gives the highest classification accuracy and set this to be the default value of the parameter in bigramBayes.

As with MP1, some of our autograder tests will use your default values for the tunable parameters inputs to bigramBayes. However, some of our tests will reset these parameter values so that we can test specific aspects of your code. In particular, your definition of bigramBayes should set the default value of pos_prior to match the development dataset and we will adjust this value for our hidden datasets.

You can continue to experiment with stemming, transforming to lowercase, and/or removing stop words. If you turn on these these features from the command line or in load_data, they will be used by bigramBayes. You can also use them by editing bigramBayes.

Making the details work

Consider Python's Counter data structure.

Use the log of the probabilities to prevent underflow/precision issues. Apply log to both sides of the equation and convert multiplication to addition. Be aware that the standard python math functions are faster than the corresponding numpy functions, when applied to individual numbers.

Zero values in the naive Bayes equations will prevent the classification from working right. Therefore, you must smooth your calculated probabilities so that they are never zero. In order to accomplish this task, use Laplace smoothing. See the lecture notes for details. The Laplace smoothing parameter α is passed as an argument to bigramBayes and you can adjust its value using the command-line arguments to mp2.py.

Tune the values of the Laplace smoothing constant using the command-line arguments to mp2.py. When you are happy with the result on the development set, edit the default values for these parameters in the definition of the function bigramBayes. Some of our tests will use your default settings and some tests will pass in new values.

You can experiment with other methods that might (or might not) improve performance. The command line options will let you transform the input words by converting them all to lowercase and/or running them through the Porter Stemmer. If you wish to turn either of these on for your autograder tests, edit the default values in the function load_data.

You could also try removing stop words from the reviews before you process them. You can add this to load_data or to the start of your bigramBayes function. You will need to find a suitable list of stop words and write a short python function to modify the input data.

No guarantees about what changes will make the accuracy better or worse. You need to figure that out by experimenting.