CS114B (Spring 2023) Homework 1
Naïve Bayes in Numpy

Due February 3, 2023

You are given `naive_bayes.py`, and `movie_reviews.zip`, the NLTK movie review corpus. Reviews are separated into a training set (80% of the data) and a development set (10% of the data). A testing set (10% of the data) has been held out and is not given to you. Within each set, reviews are sorted by sentiment (positive/negative). The files are already tokenized. Each review is in its own file. You are also given `movie_reviews_small.zip`, the toy corpus from the Jurafsky and Martin book, Exercise 4.2, in the same format. The toy corpus is described in Appendix A.

You will need to use Numpy for this assignment. A description of useful Numpy functions is in Appendix B.

Assignment

Your task is to implement a multinomial Naïve Bayes classifier using bag-of-words features and add-1 smoothing. Specifically, in `naive_bayes.py`, you should fill in the following functions:

- `train(self, train_set)`: This function should, given a folder of training documents, fill in `self.prior` and `self.likelihood`, such that:
  - `self.prior[class]` $= \log(P(\text{class}))$
  - `self.likelihood[feature, class]` $= \log(P(\text{feature}|\text{class}))$

You can use the pseudo-code given in Figure 4.2 of the Jurafsky and Martin book, reproduced below.

function TRAIN NAIVE BAYES(D, C) **returns** log $P(c)$ and log $P(w|c)$

for each class $c \in C$ # Calculate $P(c)$ terms
  N_{doc} = number of documents in D
  N_c = number of documents from D in class c
  $logprior[c] \leftarrow \log \frac{N_c}{N_{doc}}$
  $V \leftarrow$ vocabulary of D
  $bigdoc[c] \leftarrow$ **append**(d) **for** d $\in$ D **with** class c
  for each word w in V # Calculate $P(w|c)$ terms
    $count(w,c) \leftarrow$ # of occurrences of w in $bigdoc[c]$
    $loglikelihood[w,c] \leftarrow \log \frac{count(w,c) + 1}{\sum_{w' \text{ in } V} (count(w',c) + 1)}$
return *logprior*, *loglikelihood*, V

function TEST NAIVE BAYES(*testdoc*, *logprior*, *loglikelihood*, C, V) **returns** best c

for each class $c \in C$
  $sum[c] \leftarrow logprior[c]$
  for each position i in *testdoc*
    $word \leftarrow testdoc[i]$
    if $word \in V$
      $sum[c] \leftarrow sum[c] + loglikelihood[word,c]$
return $\text{argmax}_c\ sum[c]$

Figure 4.2 The naive Bayes algorithm, using add-1 smoothing. To use add-α smoothing instead, change the $+1$ to $+\alpha$ for loglikelihood counts in training.

`self.prior` and `self.likelihood` should be Numpy arrays, `self.prior` a vector (array of rank 1) of shape $(|C|,)$, and `self.likelihood` a matrix (array of rank 2) of shape $(|F|, |C|)$, where $|F|$ and $|C|$ are the numbers of features and classes, respectively. For this assignment, you should use the entire vocabulary as features. `self.feature_dict` and `self.class_dict` should be used to translate between feature/class names and indices. For example, if `self.feature_dict['couple']` is 1, and `self.class_dict['neg']` is 0, then `self.likelihood[1, 0]` should be $\log(P(\text{couple}|\text{negative}))$. You will need to fill in `self.feature_dict` and `self.class_dict` yourself. How you choose to assign features/classes to indices is up to you.

- `test(self, dev_set)`: This function should, given a folder of development (or testing) documents, return a dictionary of `results` such that:
  - `results[filename]['correct']` = correct class
  - `results[filename]['predicted']` = predicted class

  You can look at the pseudo-code in Figure 4.2 of the book for inspiration, but our procedure will be somewhat different. For each document, we will create a feature vector: if `self.feature_dict['couple']` is 1, then the second element of the vector will be the count of how many times "couple" appears in the document. We can then take the `numpy.dot` product of our vector and `self.likelihood`: the product of our feature vector of length $|F|$ and our $|F| \times |C|$ log-likelihood matrix will be a vector of length $|C|$ that contains, for each class, the log-likelihood of the document given the class. We can then add `self.prior` to this vector and take the argmax to find the most probable class.

- `evaluate(self, results)`: This function should, given the results of `test`, compute precision, recall, and F1 score for each class, as well as the overall accuracy, and print them in a readable format. Recall that (where c_{ij} is the number of documents classified as being in class i that were actually in class j):
  - $\text{precision}(i) = \dfrac{c_{ii}}{\sum_j c_{ij}}$
  - $\text{recall}(i) = \dfrac{c_{ii}}{\sum_j c_{ji}}$
  - $\text{F1}(i) = \dfrac{2 \times \text{precision}(i) \times \text{recall}(i)}{\text{precision}(i) + \text{recall}(i)}$
  - $\text{accuracy} = \dfrac{\sum_i c_{ii}}{\sum_i \sum_j c_{ij}}$

  When calculating your evaluation metrics, it may be helpful to use a confusion matrix. The `confusion_matrix` defined in `evaluate` can be populated as follows: `confusion_matrix[class_1][class_2]` = the number of documents classified as `class_1` that are actually in `class_2`.

Grading

Grades will be determined as follows:

- 20%: Any good faith effort will receive 20% as a base.
- 10%: `self.class_dict` and `self.feature_dict` are dictionaries that map class names and feature names, respectively, to indices.
- 10%: `self.prior` is a Numpy array such that `self.prior[class]` $= \log(P(\text{class}))$.
- 10%: `self.likelihood` is a Numpy array such that `self.likelihood[feature, class]` $= \log(P(\text{feature}|\text{class}))$.
- 10%: For each document, there is a feature vector of word counts in the document.
- 20%: The most probable class is found by taking the product of the feature vector and `self.likelihood`, adding `self.prior`, and taking the argmax.
- 10%: Precision, recall, F1, and accuracy are computed and printed correctly.
- 10%: There are two sets of precision/recall/F1 (one for each class), and one overall accuracy.
- Within these parameters, partial credit will be assigned where possible.

Submission Instructions

Please submit one file to LATTE: `naive_bayes.py`. You do not need to write anything up for this assignment.

Appendix A: Toy corpus

The toy corpus contains the following short movie reviews, each labeled with a genre, either comedy or action, as the training set:

document	class
fly fast shoot love	action
fun couple love love	comedy
fast furious shoot	action
couple fly fast fun fun	comedy
furious shoot shoot fun	action

and a testing document:

fast couple shoot fly

It is recommended that you create (conditional) probability tables such as those shown below. You can then use these tables to check your work.

class	action	comedy
P(class)		

P(feature|class)		class	
		action	comedy
feature	fast		
	couple		
	shoot		
	fly		

Appendix B: Useful Numpy functions

You may find the following Numpy functions useful:

- `numpy.zeros(shape)`: Returns an array of given `shape`, filled with zeros.
- `numpy.log(x)`: If `x` is a number, returns the (natural) log of `x`. If `x` is an array, returns an array containing the logs of each element of `x`.
- `numpy.dot(a, b)`: Returns the product of `a` and `b`, where the type of product depends on the shapes of `a` and `b`.
- `numpy.argmax(a, axis=None)`: Given an array `a`, returns the argmax(es) along an axis. If no axis is given, returns the argmax over the entire array (flattening it into a vector if necessary).
- `numpy.sum(a, axis=None)`: Given an array `a`, returns the sum(s) over an axis. If no axis is given, returns the sum over the entire array.
- `numpy.trace(a)`: Given a matrix `a`, returns the sum along the diagonal.