CS440/ECE448 Spring 2022

Assignment 1: Naive Bayes

Due date: Monday January 31st, End of the Day

Updated 2022 By: Xiuye Zhu, Feiyu Zhang, and Mohit Goyal
Updated 2021 By: Ayush Sarkar and Kiran Ramnath
Updated 2020 By: Jaewook Yeom
Updated 2019 By: Kedan Li and Weilin Zhang
Updated 2018 By: Justin Lizama and Medhini Narasimhan

Spam emails are a common issue we all encounter! In this assignment, you will use the Naive Bayes algorithm to tr
emails that is provided to you. The task in Part 1 is to learn a bag of words (unigram) model that will classify an em
on the words it contains. In Part 2, you will combine the unigram and bigram models to achieve better performance

Contents

General guidelines

Basic instructions are similar to MP 1:

- For general instructions, see the main MP page.
- In addition to the standard python libraries, you should import nltk and numpy.
- Code will be submitted on gradescope.

Problem Statement

You are given a dataset consisting of Spam and Ham (not spam) emails. Using the training set, you will learn a Nai
right class label given an unseen email. Use the development set to test the accuracy of your learned model, with t
have multiple hidden test sets that we will use to run your code after you turn it in.

Dataset

The dataset that we have provided to you consists of 1500 ham and 1500 spam emails, a subset of the Enron–Spa
Androutsopoulos. This dataset is split into 2000 training examples and 1000 development examples. This dataset is
template code provided. You will also find another dataset located under the `counter-data` folder of the template c
grade.py file to check whether your model implementation is correct.

Background

The bag of words model in NLP is a simple unigram model which considers a text to be represented as a bag of in
the position the words appear in, and only pay attention to their frequency in the text. Here, each email consists of

theorem, you need to compute the probability that the label of an email (Y) should be ham (Y=ham) given the word estimate the posterior probabilities:

$$P(Y = \text{ham}|\text{words}) = \frac{P(Y = \text{ham})}{P(\text{words})} \prod_{x \in \text{words in the e-mail}} P(X = x|Y = \text{ham})$$

$$P(Y = \text{spam}|\text{words}) = \frac{P(Y = \text{spam})}{P(\text{words})} \prod_{x \in \text{words in the e-mail}} P(X = x|Y = \text{span}$$

You will need to use log of the probabilities to prevent underflow/precision issues; apply log to both sides of the same in both formulas, so you can omit it (set term to 1).

Part 1: Unigram Model

Before starting: Make sure you install the nltk package with 'pip install nltk' and/or 'pip3 install nltk', depending on (we suggest you use Python 3.8 or 3.9). Otherwise, the provided code will not run. More information about the pac

- **Training Phase:** Use the training set to build a bag of words model using the emails. Note that the method **rea** which will already pre-process the training set for you, such that the training set is a list of lists of words (eac one email). The purpose of the training set is to help you calculate $P(X = x|Y = \text{ham})$ and $P(X = x|Y = \text{spa}$ phase.

  Hint: Think about how you could use the training data to help you calculate $P(X = x|Y = \text{ham})$ and $P(X = x|$ (testing) phase. Note that $P(X = \text{tiger}|Y = \text{ham})$ is the probability that any given word, in a ham e-mail, is th you should be able to compute $P(X = x|Y = \text{ham})$ and $P(X = x|Y = \text{spam})$ for any word. Also, look into usi things easier for you coding-wise.

- **Development Phase:** In the development phase, you will calculate the $P(Y = \text{ham}|\text{words})$ and $P(Y = \text{spam}|\text{w}$ development set. You will classify each email in the development set as a ham or spam email depending on wh value. You should return a list containing labels for each of the emails in the development set (label order shou in the given development set, so we can grade correctly). Note that your code should use only the training set not use the development data or any external sources of information.

  Hint: Note that the prior probabilities will already be known (since you will specify the positive prior probability remember that you can simply omit $P(\text{words})$ by setting it to 1. Then, your only remaining task is: for each doc $P(X = x|Y = \text{ham})$ and $P(X = x|Y = \text{spam})$ for each of the words. After that, you will be able to compute P $P(Y = \text{spam}|\text{words})$ for each email in the development set using the formulas above.

Laplace Smoothing: You will need to make sure you smooth the likelihoods to prevent zero probabilities. In order t smoothing. This is where the Laplace smoothing parameter will come into play. You can use the following formula the likelihood probabilities:

$$P(X = x|Y = y) = \frac{\text{Count}(X = x|Y = y) + k}{N(Y = y) + k(1 + |X|_{Y=y})}$$

where Count(X=x|Y=y) is the number of times that the word was x given that the label was y, k is the Laplace smoo $N(Y = y) = \sum_x \text{Count}(X = x|Y = y)$ is the total number of word tokens that exist in e-mails that are labeled Y= word types that exist in e-mails labeled Y=y. **A note on out of vocabulary (OOV) words:** Suppose that the dev set in the training set. In that case, the term $\text{Count}(X = x|Y = y) = 0$, but all of the other terms in the above formula This should not be something you compute during your analysis of the training set, but it's a backoff value that you analyze the dev set.

Part 2: Unigram and Bigram Models

For Part 2, you will implement the naive Bayes algorithm over a bigram model (as opposed to a unigram model like bigram model and the unigram model into a mixture model defined with parameter λ:

$$P(Y = \text{Spam}|\text{email}) = \left(P(\text{Y} = \text{Spam}) \prod_{x_i \in \text{unigrams in the email}} P(X = x_i|Y = \text{Spam}) \right)^{1-\lambda} \left(P(Y = \text{Spam}) \prod_{b_i \in \text{bigrams ir}} \right.$$

You should not implement the equation above directly -- you should implement its logarithm.

You can choose to find the best parameter λ that gives the highest classification accuracy. There are additional pa details, see here). **However, I should note that you can get away with not tuning these parameters at all, since th for you when you grade. So feel free to play around with these parameters, but in the end, the hyperparameters local machine won't matter in grading.**

Some weird things happen when using the bigram model. One of them happens when you use Laplace smoothing t from training set data. Obviously, $\text{Count}(\text{Bigram}_i = b_i|Y = \text{spam})$ is the number of times bigram b_i occurred in s Laplace smoothing coefficient. The weird thing is that $N(Y = \text{spam})$ should equal the total number of bigrams tha total number of unigrams that occurred in spam emails. Similarly, $|X|_{Y=\text{spam}}$ should equal the total number of distir emails, **plus** the total number of distinct unigrams that occurred in spam emails. We're not sure why the bigram mc unigrams counted in the denominator, but for some reason, it does.

Part 3: Extra credit

For the extra credit part of this MP, you can try a more informative measure of the importance of each word. tf-id text mining applications to determine how important a word is to a document within a collection of documents. Th how frequently a word appears in a document (thus tf is kind of related to naive Bayes, though it's not exactly the frequency) term determines how rare (and thus perhaps more informative) a word is in the collection. A high tf-idf high term frequency in a particular document, and also that it is has a low document frequency (low number of doc

Like a bag-of-words, tf-idf can be used in classifying documents. For the extra credit portion, you will implement and 2. However, since the extra credit portion is worth only 10%, we will ask you to complete an easier task than ir classifying the documents, you will find the word with the highest tf-idf value from each of the documents in the d list of these words (with the highest tf-idf values).

For this MP, you should treat the entire training dataset (both ham and spam documents) as the collection of docu term. Your idf term for any given word should be calculated based on only the training dataset, not the developmer calculate your tf term based on the term frequency in the document you are evaluating from the development set. obtain results consistent with the autograder.

There are multiple ways in which tf-idf gets implemented in practice. For consistency, we will use the following for word:

$$\text{tf-idf}(\text{word } w, \text{document } A) = \frac{(\#\text{ of times word w appears in doc. A})}{(\text{total \# of words in doc. A})} \cdot \log\left(\frac{\text{total \# of do}}{1 + \#\text{ of docs in train}}\right.$$

If there are terms with the same tf-idf value within the same document, choose the first term for tie-breaking.

Sidenote: A cursory look at the words would show you the importance of governing the outputs of such methods b that's beyond the scope of this assignment. Interested students could look up literature on bias and fairness in AI.

Provided Code Skeleton

We have provided (tar, zip) all the code to get you started on your MP.

Note that the only files you should need to modify are **naive_bayes.py** for Part 1 and **tf-idf.py** for Part 2. Those are the autograder.

Files Used for Part 1&2

- **reader.py** – This file is responsible for reading in the data set. It takes in all of the emails, splits each of those if you used the --stemming flag, and then stores all of the lists of words into a larger list (so that each individ

single email). Note that this file is used for both parts of the assignment (Part 1 and Part 2).
- **mp1.py** – This is the interactive file that starts the program for Part 1, and computes the accuracy, precision, r implementation of naive Bayes.
- **grade.py** – This is a test-grader. It will grade your homework using the provided development test data. The a same grading, and also grades some hidden tests.
- **naive_bayes.py** – This is the file where you will be doing all of your work for Part 1. The function naiveBayes() labels, development set, smoothing parameter, and positive prior probability. The training data provided is the labels is the list of labels corresponding to each email in the training data. The development set is the list of e implementation on. The smoothing parameter is the laplace smoothing parameter you specified with --laplace probability is a value between 0 and 1 you specified with --pos_prior. You will have naiveBayes() output the pr from your Naive Bayes model.

Do not modify the provided code. You will only have to modify **naive_bayes.py** for Part 1.

mp1.py

Here is an example of how you would run your code for Part 1 from your terminal/shell:

```
python3 mp1.py --training ../data/spam_data/train --development ../data/spam_data/dev --s
--laplace 1.0 --pos_prior 0.8
```

Here is an example of how you would run your code for Part 2 from your terminal/shell:

```
python3 mp1.py --bigram True --training ../data/spam_data/train --development ../data/spa
lowercase False --bigram_laplace 1.0 --bigram_lambda 0.5 --pos_prior 0.8
```

Note that you can and should change the parameters as necessary. To understand more about how to run the MP your terminal.

Optional: How to change your flag

We provide some flags for you to play around when running your code (for help on how to run the code, see here).

For example, you can use the --lower_case flag to cast all words into lower case. You can also tune the laplace sr laplace flag. You can also tune the --stemming and --pos_prior parameters. You should be able to boost the moc parameters. **However, note that when we grade your MP, we will use our own flags (stemming, lower_case, pos_**

grade.py

This file, similar to MP1, is for you to be able to evaluate whether or not you have implemented the unigram and mi consists of three fundamental tests:

- It checks if your mixture model approach is correct (whether or not your model is actually a mixture of unigram located under bigram_check
- It checks if your unigram model is able to reach the correct accuracy threshold for the development set provid
- It checks if your mixture model is able to reach the correct accuracy threshold for the development set provide

Unlike the grade.py file provided to you in MP1, this version provides you with only a subset of the tests that your c file should at least give you a general idea on whether or not your implementations are correct. Your code will go t provided to you once it is submitted to gradescope. Here is an example of how you would run grade.py from your t

```
python3 grade.py
```

Files Used for Extra Credit

- **reader.py** – This is the same file used in Part 1 and Part 2 (see above for description for this file).
- **run_tf_idf.py** – This is the main file that lets you run the program for the Extra Credit Part.

- **tf_idf.py** – (this is the only file you need to edit) This is the file where you will be doing all of your work for the compute_tf_idf() takes as input the training data, training labels, and development set. The training data provic training labels is the list of labels corresponding to each email in the training data. The development set is the extract the words with the highest tf-idf values. You should return a list (with size equal to that of dev_set) tha set with the highest tf-idf values. More specifically, the list should contain the word with the highest tf-idf valu development set. The order of the words in the list returned should correspond to the order of documents in th
- **grade_tfidf.py** – This file allows you to test your code on the development test provided. We will test your coc set should give you a general sense of whether your implementation is correct or not.

Do not modify the provided code. You will only have to modify **tf_idf.py** for the Extra Credit Part. Here is an examp the Extra Credit Part from your terminal/shell:

```
python3 run_tests_tfidf.py --training data/spam_data/train --development data/spam_data/d
```

What to Submit

Submit only your **naive_bayes.py** file to Gradescope, under the assignment called **MP1.**

If you completed the extra credit portion, submit only the **tf_idf.py** file to Gradescope, under the assignment called