Spam filtering using naïve Bayes and perceptron classifiers

CSCI 544, Spring 2021: Assignment 1
Due Date: March 8th, 4pm.

Introduction

The goal of this assignment is to get some experience implementing the simple but effective machine learning techniques, naïve Bayes classification and perceptron classification, and applying them to a binary text classification task (i.e., spam detection). This assignment will describe the details of particular implementations of a naïve Bayes classifier and a perceptron classifier with the categories spam and ham (i.e., not spam). These implementations are exactly the same as the implementations discussed in class (naïve Bayes with add-one smoothing and averaged perceptron). The teaching assistants have created solutions (one for naïve Bayes with add-one smoothing and one for averaged perceptron) following these same directions, and your work will be evaluated based on its performance on an unseen test set in comparison to these solutions. In addition, we ask you to submit a short report including the following information: your performance on the development data.

This is also an introduction to the Vocareum platform. Vocareum is like Blackboard for programming assignments. You will use Vocareum to submit your assignment, and later to check your grade. You can submit your assignment as many times as you like, although we will grade the last version submitted applying any appropriate late penalties. So only submit after the deadline if you think your new version will result in a better grade after the late penalty. Each time you submit your code, Vocareum will run it, perform some basic checks, and tell you the results on the development data. Note however that your grade will be computed based on the performance of your implementations on the test data (unseen for you).

We have uploaded the email data for this assignment to Vocareum. There is training data, development data, and test data. The test data cannot be seen by you. When you submit your assignment, Vocareum will run your nblearn.py and perclearn.py scripts (see below) on the entire training set, and then run your nbclassify.py and percclassify.py scripts (also described below) on the development data. Vocareum will alert you to any errors and report your performance on the development data. The training and development data are also available for download via Blackboard, and you will need to work locally on your own computer to run the experiments described here. However, the majority of your grade will depend on how your code performs on Vocareum, not your computer. You will be writing your code in Python3. Vocareum is currently using version 3.7.5 of Python. You should use only default Python libraries and NumPy. No other external Python libraries are permitted.

Naïve Bayes Classifier in Python

You will write three programs: **nblearn.py** will learn a naïve Bayes model from labeled data, **nbclassify.py** will use the model to classify new data, and **nbevaluate.py** will print precision, recall, and F1 scores based on the output of nbclassify.py on development (i.e., labeled) data. In addition, you will run experiments using these programs and report the results using this template:
https://kgeorgila.github.io/teaching/cs544-spring2021/assignment1/report.txt
Note that you should report your results for both naïve Bayes and perceptron in one report.txt file.

1. Write nblearn.py
nblearn.py will be invoked in the following way:

```
>python3 nblearn.py input_data_path
```
and output a model file called nbmodel.txt

1.1 Reading data
The argument is a data directory. The script should search through the directory recursively looking for subdirectories containing the folders: "ham" and "spam". Note that there can be multiple "ham" and "spam" folders in the data directory. Emails are stored in files with the extension ".txt" under these directories. The file structure of the training and development data on Blackboard is exactly the same as the data on Vocareum although these directories (i.e., "dev" and "train") will be located in different places on Vocareum and on your personal computer.

ham and **spam** folders contain emails failing into the category of the folder name (i.e., a spam folder will contain only spam emails and a ham folder will contain only ham emails). Each email is stored in a separate text file. The emails have been preprocessed removing HTML tags, and leaving only the body and the subject. The files have been tokenized such that white space always separates tokens. Note, in the subject line, "Subject:" is considered as one token. Because of special characters in the corpus, you should use the following command to read files:
```
open(filename, "r", encoding="latin1")
```
The official solution uses all tokens. It does not remove punctuation or stop words.

1.2 Learning the model
You will need to estimate and store P(spam) and P(ham) as well as conditional probabilities P(token|spam) and P(token|ham) for all unique tokens. These probabilities should be stored in the model file **nbmodel.txt**. The format of the file is up to you but your nbclassify.py program must be able to read it.

For smoothing you should use add-one smoothing. During testing, you can simply ignore unknown tokens not seen in training (i.e., pretend they did not occur).

2. Write nbclassify.py
nbclassify.py will be invoked in the following way:
```
>python3 nbclassify.py input_data_path
```

The argument is again a data directory but you should not make any assumptions about the structure of the directory. Instead, you should search the directory for files with the extension ".txt". nbclassify.py should read the parameters of the naïve Bayes model from the file **nbmodel.txt**, and classify each ".txt" file in the data directory as "ham" or "spam", and write the result to a text file called **nboutput.txt** in the format below:
```
LABEL path_1
LABEL path_2
⋮
```

In the above format, LABEL is either "spam" or "ham" and path is the absolute path to the file including the filename (e.g., on Windows a path might be: "C:\dev\4\ham\0026.2000-01-17.beck.ham.txt"). nbclassify.py should ignore any unknown tokens not seen in training (i.e., pretend they did not occur).

3. Write nbevaluate.py
nbevaluate.py will be invoked in the following way:

```
>python3 nbevaluate.py nboutput_filename
```

nboutput_filename is the output file of nbclassify.py described above. For each line in the file, nbevaluate.py will split the line into the guessed label and file path. nbevaluate.py will search for ham or spam in the path to determine the true label of the example (i.e., “spam” or “ham”). If neither is found, then it will skip to the next line in the file. Otherwise, the true label will be compared to the guessed label. You will need to maintain counts allowing you to calculate precision, recall, and F1 score for both spam and ham and print them once all output has been processed. The values for precision, recall, and F1 score should be in the range between 0 and 1 (not percentages). You will include these values in your report. Note that your program should not crash if no labeled examples are seen and you should be careful to avoid dividing by zero.

4. Run experiments and report results using template:
https://kgeorgila.github.io/teaching/cs544-spring2021/assignment1/report.txt
You should report your results for both naïve Bayes and perceptron in one report.txt file.

The experiment is simply to

- run nblearn.py on the entire training data,
- run nbclassify.py on the dev data using the resulting model
- use nbevaluate.py to measure performance and add the results to your report.

Note that accuracy as well as precision, recall, and F1 score for each class (on the dev data) will be reported in Vocareum when you submit your assignment allowing you to check your work.

Perceptron Classifier in Python

You will write three programs: **perclearn.py** will learn a perceptron model from labeled data, **percclassify.py** will use the model to classify new data, and **percevaluate.py** will print precision, recall, and F1 scores based on the output of percclassify.py on development (i.e., labeled) data. In addition, you will run experiments using these programs and report the results using this template:
https://kgeorgila.github.io/teaching/cs544-spring2021/assignment1/report.txt
Note that you should report your results for both naïve Bayes and perceptron in one report.txt file.

1. Write perclearn.py
perclearn.py will be invoked in the following way:
```
>python3 perclearn.py input_data_path
```
and output a model file called percmodel.txt

1.1 Reading data
The argument is a data directory. The script should search through the directory recursively looking for subdirectories containing the folders: "ham" and "spam". Note that there can be multiple "ham" and "spam" folders in the data directory. Emails are stored in files with the extension ".txt" under these directories. The file structure of the training and development data on Blackboard is exactly the same as the data on Vocareum although these directories (i.e., "dev" and "train") will be located in different places on Vocareum and on your personal computer.

ham and **spam** folders contain emails failing into the category of the folder name (i.e., a spam folder will contain only spam emails and a ham folder will contain only ham emails). Each email is stored in a separate text file. The emails have been preprocessed removing HTML tags, and leaving only the body

and the subject. The files have been tokenized such that white space always separates tokens. Note, in the subject line, "Subject:" is considered as one token. Because of special characters in the corpus, you should use the following command to read files:
open(filename, "r", encoding="latin1")
The official solution uses all tokens. It does not remove punctuation or stop words.

1.2 Learning the model
You will need to estimate the perceptron weights and store them in the model file **percmodel.txt**. The format of the file is up to you but your percclassify.py program must be able to read it.

You should train the model for 100 iterations. You may or not shuffle the data in each iteration, it is up to you. The official solution will not use shuffling. During testing, you can simply ignore unknown tokens not seen in training (i.e., pretend they did not occur).

2. Write percclassify.py
percclassify.py will be invoked in the following way:
>python3 percclassify.py input_data_path

The argument is again a data directory but you should not make any assumptions about the structure of the directory. Instead, you should search the directory for files with the extension ".txt". percclassify.py should read the parameters of the perceptron model from the file **percmodel.txt**, and classify each ".txt" file in the data directory as "ham" or "spam", and write the result to a text file called **percoutput.txt** in the format below:
LABEL path_1
LABEL path_2
⋮

In the above format, LABEL is either “spam” or “ham” and path is the absolute path to the file including the filename (e.g., on Windows a path might be: "C:\dev\4\ham\0026.2000-01-17.beck.ham.txt"). percclassify.py should ignore any unknown tokens not seen in training (i.e., pretend they did not occur).

3. Write percevaluate.py
percevaluate.py will be invoked in the following way:
>python3 percevaluate.py percoutput_filename

percoutput_filename is the output file of percclassify.py described above. For each line in the file, percevaluate.py will split the line into the guessed label and file path. percevaluate.py will search for ham or spam in the path to determine the true label of the example (i.e., “spam” or “ham”). If neither is found, then it will skip to the next line in the file. Otherwise, the true label will be compared to the guessed label. You will need to maintain counts allowing you to calculate precision, recall, and F1 score for both spam and ham and print them once all output has been processed. The values for precision, recall, and F1 score should be in the range between 0 and 1 (not percentages). You will include these values in your report. Note that your program should not crash if no labeled examples are seen and you should be careful to avoid dividing by zero.

4. Run experiments and report results using template:
https://kgeorgila.github.io/teaching/cs544-spring2021/assignment1/report.txt
You should report your results for both naïve Bayes and perceptron in one report.txt file.

The experiment is simply to

- run perclearn.py on the entire training data,
- run percclassify.py on the dev data using the resulting model
- use percevaluate.py to measure performance and add the results to your report.

Note that accuracy as well as precision, recall, and F1 score for each class (on the dev data) will be reported in Vocareum when you submit your assignment allowing you to check your work.

What to turn in and Grading

You need to submit the following on Vocareum:

- **report.txt**. Download and fill in the template (https://kgeorgila.github.io/teaching/cs544-spring2021/assignment1/report.txt). Include the results from your experiments and make sure that none of your code writes to a file called report.txt. We don't want to accidently damage the file when we test your code.
- **All your code**: nblearn.py, nbclassify.py, nbevaluate.py, perclearn.py, percclassify.py, percevaluate.py, and any additional code you created for the assignment. This is required and the academic honesty policy (see rules below) applies to all your code.

12% of your grade (12 points, 1 point for each question) is based on the report. Make sure to answer every question. The other 88% is based on the performance (accuracy) of your code **on the test data** compared to the official solutions written by the teaching assistants following the directions above. If your performance is the same or better then you receive full credit (44 points for naïve Bayes and 44 points for perceptron). Otherwise, your score is proportional to your relative performance:
For naïve Bayes
naïve_Bayes_score = 44 points * your_nb_accuracy / accuracy_of_nb_solution
For perceptron
perceptron_score = 44 points * your_perc_accuracy / accuracy_of_perc_solution
final_score = report_score + naïve_Bayes_score + perceptron_score

Late Policy

- On time (no penalty): before March 8th, 4pm.
- One day late (10 point penalty): before March 9th, 4pm.
- Two days late (20 point penalty): before March 10th, 4pm.
- Three days late (30 point penalty): before March 11th, 4pm.
- Zero credit after March 11th, 4pm.

Other Rules

- DO NOT look for any kind of help on the web outside of the Python documentation.
- DO NOT use any external libraries other than the default Python libraries and the NumPy library.
- Questions should go to Piazza first, not email. This lets any of the TAs or the instructor answer, and lets all students see the answer.
- When posting to Piazza, please check first that your question has not been answered in another thread.
- DO NOT wait until the last minute to work on the assignment. You may get stuck and last minute Piazza posts may not be answered in time.

- This is an individual assignment. DO NOT work in teams or collaborate with others. You must be the sole author of 100% of the code and writing that you turn in.
- DO NOT use code you find online or anywhere else.
- DO NOT post your code online or anywhere else.
- DO NOT turn in material you did not create.
- All cases of cheating or academic dishonesty will be dealt with according to University policy.

FAQ

a) **How will the TAs/Professor grade my HW?**
We have written scripts which will run on Vocareum. Since we automate the process of grading, make sure that you write your code to match the output format "exactly". If you do not do this there will be a penalty.

b) **I found a piece of code on the web. Is it ok to use it?**
No! We run plagiarism checks on the code you write. The plagiarism detector is a smart algorithm which can tell if you've copied the code from elsewhere/others.
Instead please contact TAs/Professor during the office hours with your questions.

c) **Vocareum terminates my code before it finishes, but it finishes executing on my computer.**
This usually means that your implementation is inefficient. Keep an eye out for places where you iterate. Run time issues can be solved by using efficient data structures (Hashmap, Hashset, etc.).

d) **My code works on my computer but not on Vocareum. Do I get an extension?**
The late policy above still applies, and is automatically enforced by the system. Submit your code incrementally to make sure it functions on Vocareum.

e) **When are the office hours ?**
Please refer to the course website: https://kgeorgila.github.io/teaching/cs544-spring2021/index.html and Blackboard for the Zoom links.