CSCI544, Fall 2016: Assignment 1

Due Date: September 23rd, 4pm.

Introduction

The goal of this assignment is to get some experience implementing the simple but effective machine learning technique, Naïve Bayes classification, and applying it to a binary text classification task (i.e., spam detection). This assignment will describe the details of a particular implementation of a Naïve Bayes classifier with the categories spam and ham (i.e., not spam). The teaching assistants have created a solution following these same directions, and your work will be evaluated based on its performance on an unseen test set in comparison to this solution. In addition, we ask you to submit a short report including the following information:

- Your performance on the development data.
- Your performance on the development data with a model built from only 10% of the training data.
- Your performance on the development data after trying a different approach to smoothing, common words and/or unknown words.

This is also an introduction to the Vocareum platform which we will use for all the individual assignments. Vocareum is like BlackBoard for programming assignments. You will use Vocareum to submit your assignments, 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.

We have uploaded the email data for this assignment to Vocareum. There is training data, development data and test data. When you submit your assignment, Vocareum will run your nblearn.py script (see below) on the entire training set, and then run your nbclassify.py script (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 some of 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 3.4.3 of Python.

Naïve Bayes Classifier in Python

You will write two programs: **nblearn.py** will learn a naive Bayes model from labeled data, and **nbclassify.py** will use the model to classify new data. nblearn.py will be invoked in the following way:

```
>python3 nblearn.py /path/to/input
```

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")
```

For this assignment, you are required to use tokens as features. You are free to use any smoothing method, but you should at least use add-one smoothing. The official solution will use add-one smoothing. During testing, you can simply ignore unknown tokens not seen in training (i.e., pretend they did not occur).

nblearn.py will learn a naive Bayes model from the training data, and write the model parameters to a file called **nbmodel.txt**. The format of the model is up to you, but it should contain sufficient information for nbclassify.py to successfully classify new data.

nbclassify.py will be invoked in the following way:

```
>python3 nbclassify.py /path/to/input
```

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 a naive 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 described on the next page.

LABEL path_1
LABEL path_2
⋮

In the above format, LABEL is either “spam” or “ham” and path is the path to the file, and the name of filename (e.g., on Windows a path might be: "C:\dev\4\ham\0001.2000-01-17.beck.ham.txt").

To write your report, you will need to do some additional work. First, you will need to be able to calculate precision, recall and F1 score assuming that you run nbclassify.py on a labeled corpus (i.e., a data directory with "ham" and "spam" subdirectories). You can write additional Python script(s) or modify nbclassify.py to include this functionality. However, nbclassify.py must still work as shown above so that we can test (i.e., we should be able to run it by simply including the data directory as the sole command line argument, and it should still work even if there are no "ham" and "spam" subdirectories).

Second, you will need a way to train a model with 10% of the training data. You need to be careful how you divide the data (e.g., it would be a disaster if your training data only had one class). We recommend calculating a target number of files (i.e., 10% of the total) and selecting half of those files to be ham examples, and half to be spam examples. You can do this manually (e.g., copying files to create a smaller training directory) or by writing Python code (i.e., separate scripts or modifying nblearn.py). If you modify nblearn.py, it must still work as shown above so that we can test your code (i.e., we should be able to run it by simply including the data directory as the sole command line argument, and by default it should use all the data for training the model).

Your third task is to attempt at least one modification of the classifier described above such as changing smoothing, treatment of common words, or treatment of unknown words. We strongly recommend that the modification is NOT the default behavior of nbclassify.py as the modification could hurt performance on the unseen test set. Also, remember that we need to be able to run nbclassify.py as described above with the data directory being the only command line argument.

What to turn in and Grading

You need to submit the following on Vocareum:

- **Report.txt**. Download and fill in the template from the class website. Include the results from your experiments as well as a description of the modifications you used to try to improve performance on the development set. 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 and any additional code you created for the assignment (e.g., to calculate performance scores). This is required and the academic honesty policy (see rules below) applies to all your code.

10% of your grade (10 points) is based on the report. Make sure to answer every question. The other 90% is based on the performance of your code compared to the official solution written by the teaching

assistants following the directions above. If your performance is the same or better then you receive full credit. Otherwise, your score is proportional to your relative performance:

90 points * (1 – (performance_of_solution – your_performance) / performance_of_solution)

Late Policy

- On time (no penalty): before September 23rd, 4pm.
- One day late (10% penalty): before September 24th, 4pm.
- Two days late (20% penalty): before September 25th, 4pm.
- Three days late (30% penalty): before September 26th, 4pm.
- Zero credit after September 26th, 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.
- Questions should go to Piazza first not email. This lets any of the TAs or instructors 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 turn in material you did not create.
- All cases of cheating or academic dishonesty will be dealt with according to University policy.

FAQ

a) **The code runs on my computer but not on Vocareum.**
Vocareum has been tested for the same course in the previous terms and has been shown to work well. If your code doesn't run then there are some issues with your code, and you are responsible for correcting them. Test your code by submitting it on Vocareum incrementally and make sure it works.

b) How will the TAs/Professors grade my HW?

We will be writing 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. You have been warned!

c) 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. If you're caught copying the code, strict measures will be enforced. No exceptions!

Please contact TAs/Professors during the office hours with your questions and we will make sure you understand the concepts and avoid unpleasant circumstances.

d) 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.).

e) I assumed my code will run on Vocareum, but I wasn't proactive to test my code earlier. I couldn't meet the deadline because of this.

The late policy above still applies, and is automatically enforced by the system. Submit your code incrementally to make sure it functions on Vocareum. You have been warned!

f) How do I log into Vocareum ?

You will receive an email with instructions. Sometimes, if you have been a part of a course where Vocareum was used previously, you will see the course listed once you log in. Please send the TAs an email if you have problems.

g) When are the office hours ?

Please refer to the course website. http://nld.ict.usc.edu/cs544-fall2016/