Carnegie Mellon University

10601a: Homework #9 - Naive Bayes

TAs-in-charge:
Richard Fan (rmfan@andrew.cmu.edu)

Assigned: 9 November 2016.
Due: 11:59:59pm on 15 November 2016.
Late Penalty: 20% per day.

	Gates 5th Floor Commons	Table
Thr	18:00-19:00	TBD
Sat	14:00-15:00	TBD
Sun	19:00-21:00	TBD
Mon	16:00-17:00	TBD
Tue	17:00-18:30	TBD

Policy on Collaboration among Students

The purpose of student collaboration is to facilitate learning, not to circumvent it. Studying the material in groups is strongly encouraged. It is also allowed to seek help from other students in understanding the material needed to solve a particular homework problem, provided no written notes are shared, or are taken at that time, and provided learning is facilitated, not circumvented. **The actual solution must be done by each student alone**.

The purpose of programming assignments in this course is to make sure you truly understand the relevant techniques. **All code must be written by each student alone**. We will strictly enforce this policy, by carefully inspecting your code using sophisticated detection techniques. You have been warned!

The presence or absence of any form of help or collaboration, whether given or received, must be explicitly stated and disclosed in full by all involved. Specifically, **each assignment must contain a file named collaboration.txt where you will answer the following questions**:

- Did you receive any help whatsoever from anyone in solving this assignment? Yes / No. If you answered 'yes', give full details? (e.g."Jane explained to me what is asked in Question 3.4").
- Did you give any help whatsoever to anyone in solving this assignment? Yes / No. If you answered 'yes', give full details? (e.g. "I pointed Joe to section 2.3 to help him with Question 2").
- Did you find or come across code that implements any part of this assignment ? Yes / No. If you answered 'yes', give full details? (e.g. book & page, URL & location within the page, etc)

If you gave help after turning in your own assignment and/or after answering the questions above, you must update your answers before the assignment's deadline, if necessary by emailing the TA in charge of the assignment.

You are encouraged to read books and other instructional materials, both online and offline, to help you understand the concepts and algorithms taught in class. These materials may contain example code or pseudo code, which may help you better understand an algorithm or an implementation detail. However, when you implement your own solution to an assignment, you must put all materials aside, and write your code **completely on your own, starting "from scratch"**. Specifically, you may not use any code you found or came across. If you find or come across code that implements any part of your assignment, you must disclose this fact in your collaboration statement.

Some of the homework assignments used in this class may have been used in prior versions of this class, or in classes at other institutions, or elsewhere. Avoiding the use of heavily tested assignments will detract from the main purpose of these assignments, which is to reinforce the material and stimulate thinking. Because some of these assignments may have been used before, solutions to them may be, or may have been, available online, or from other people or sources. **It is explicitly forbidden to use any such sources, or to consult people who have solved these problems before. It is explicitly forbidden to search for these problems or their solutions on the internet.** You must solve the homework assignments completely on your own. For programming assignments, this means you must write your programs completely by yourself, and not use any code from any source whatsoever. I will be actively monitoring your compliance, and any violation will be dealt with harshly. Collaboration with other students who are currently taking the class is allowed, but only under the conditions stated above. Collaboration without full disclosure will be handled severely, in compliance with CMU's Policy on Cheating and Plagiarism. All violations (even first one) of course policies will always be reported to the university authorities, will carry severe penalties, usually failure in the course, and can even lead to dismissal from the university. You have been warned!

0 General Instructions

In this homework you have to choose Python, Java, C or C++ as your programming language, and you must respect that choice for all questions in this homework. The autolab environment has **Python 2.7** and **Java 7** (maybe higher) installed—check your code runs on linux.andrew.cmu.edu.

1 Building a Naive Bayes Classifier

Your task for this problem is to implement a Naive Bayes classifier to classify a political blog as being “liberal” or “conservative”. After implementing the basic algorithm, you will be asked to extend it in a few ways and analyze its behavior. The data set to be used for this assignment is a set of self-identified liberal and conservative blogs[1]. We have rearranged and preprocessed the data slightly so as to make it easier to do this assignment. The data set is available on autolab. The dataset contains a total of 120 blogs, among which 56 were identified by their author as being “liberal”, with the remaining 64 considered by their author to be “conservative”. Each blog is stored in a separate file, within which each line is a separate word. Files with a name of the form “lib*.txt” are liberal blogs, and files with a name of the form “con*.txt” are conservative. The files split.train and split.test contain lists of files to be used for training and testing a classifier. (All files are UNIX plain text.)

Implement the Naive Bayes algorithm, using smoothing, as shown in Table 6.2 on page 183 of Tom Mitchell's textbook. Your program, named as `nb.{py|java|c|c++}`, will take a set of labeled training examples in split.train and a set of test examples split.test, and classify them using a Naive Bayes classifier. You can use python or Java. Assume that all data files are in the same directory as your program. Don't upload the data when you upload your program to autolab. Your program should output predicted labels for the test data, one per line, in the order they are listed in split.test, and calculate the accuracy on the test dataset. You should ignore case—treat “President” and “president” as the same word type (transfer all the read-in words into lowercase should generally solve this problem and facilitate your output for the following "top words" and "log odd" sections). Do not worry about any other type of preprocessing—leave non-alphabetic symbols as they are.

```
$ python nb.py split.train split.test
L
L
```

[1]http://politicalbloglistings.blogspot.com/, accessed 31 October 2008

```
...
L
C
C
L
Accuracy: 0.8056
```

Use 4 digits after the decimal place, (i.e. use "%.04f").

Tips:

- Before you start coding, take a look at all the questions below so you can think about how you want to structure your code.
- Beware of numerical underflow due to products of very small numbers. If $p_1 = 1 \times 10^{-10}$, $p_2 = 2 \times 10^{-20}$ and $p_3 = 1 \times 10^{-30}$ then $p_1 \cdot p_2 \cdot p_3 = 2 \times 10^{-60}$. In Java, the minimum value a float can take is 1.4×10^{-45}, so if you simply multiplied very small floats together you would get 0, which would be very bad. A simple solution is to transform probabilities with log and use addition in place of multiplication: $\log(p_1 \cdot p_2 \cdot p_3) = \log(p_1) + \log(p_2) + \log(p_3)$. If you need to report a probability, use exponentiation.

2 Interpreting the output

Write a program **`topwords.{py|java|c|cpp}`** to take a training dataset as input and print out the top 20 words (all in lowercase) with the highest word probabilities in the liberal category as well as in the conservative category (i.e., $\hat{p}(w|C_{lib})$ and $\hat{p}(w|C_{cons})$, where C_{lib} is the liberal class and C_{cons} is the conservative class). As in the previous question, your calculations *should* use smoothing. The format should be one word per line, sorted with the highest probability first. Use the same smoothing as in Question 1 above. Print the probabilities with 4 digits after the decimal place (i.e. use "%.04f"). Output the top 20 liberal words and probabilities first, then print a blank line, then print the top 20 conservative words and probabilities:

```
$ python topwords.py split.train
liberalword1 0.0911
liberalword2 0.0505
...
liberalword20 0.0011

conservativeword1 0.1013
conservativeword2 0.0905
...
conservativeword20 0.0021
```

In topwords.txt, answer the following question: Do the two lists look different? Are there any overlapping words? In general, what kind of words are they?

3 Stop words

It is general practice to preprocess datasets and remove words like “the”, “a”, “of”, etc. before training a classifier. Rather than prespecifying a list of stop words, we can simply exclude the N most frequent words. Write a new classifier `nbStopWords.{py|java|c|cpp}` based on `nb.{py|java|c|cpp}` which additionally takes a parameter N and excludes the N most frequent words from its vocabulary before training the classifier. Here is the syntax for $N = 10$; the output should look like the output from nb.py:

```
python nbStopWords.py split.train split.test 10
```

Investigate various settings for N—what values seem to improve the classifier?—and put your observations in nbStopWords.txt.

4 Smoothing

As discussed in section 6.9.1.1 on page 179 of Mitchell, estimating probabilities based on observed fractions can cause problems when observed counts are small or 0. We will investigate various approaches to this. Following Mitchell’s notation on p. 179 and p. 183 we have:

$$P(w_k \mid v_j) \leftarrow \frac{n_k + 1}{n + |\text{Vocabulary}|}$$

In this question, we will consider:

$$P(w_k \mid v_j) \leftarrow \frac{n_k + q}{n + q \cdot |\text{Vocabulary}|}$$

Write a program `smoothing.{py|java|c|c++}` based on `nb.{py|java|c|c++}` which additionally takes a single parameter q.

Here is the syntax for $q = 1$; the output should be identical to the output from nb.py:

```
python smoothing.py split.train split.test 1
```

Try your program with $q = 0, 0.1, 0.5, 1, 5$—what values seem to improve the classifier?—and put your observations in smoothing.txt.

5 Log Odds

Write a program `topwordsLogOdds.{py|java|c|c++}` based on `topwords.{py|java|c|c++}` to print out the top 20 words (all in lower case) with the highest log-odds ratio for each class, i.e. $\log\frac{\hat{p}(w|C_{lib})}{\hat{p}(w|C_{cons})}$ and $\log\frac{\hat{p}(w|C_{cons})}{\hat{p}(w|C_{lib})}$. Assume the same input and output format as in topwords.py. Print the log-odds with 4 digits after the decimal place (i.e. use “%.04f”). Use natural log (log base e):

```
python topwordsLogOdds.py split.train
```

In topwordsLogOdds.txt answer the following: What kind of words did you find? Are there any overlapping words between the two lists? How are these words different from what you found with topwords?

6 Naive Bayes Assumption

The following section will use a toy dataset `4-features.{train|test}` with 4 binary features and a binary label. The data will have the following format:

```
feature_1 feature_2 feature_3 feature_4 label
```

Each of the feature values and labels will be 0 or 1. Feel free to manually inspect `4-features.train`.

Furthermore, **this section will not be autograded**, so you do not need to be as precise with formatting and fileIO.

Implement a Naive Bayes model for this dataset in `nb_assumptions.{py|java|c|cpp}`. For feature value x_k and label y_i, you should calculate $P(x_k|y_i)$ as the fraction of datapoints with label y_i that have the feature value x_k. You should be able to heavily reuse the code from `nb.{py|java|c|cpp}` for this section.

Given a training file `4-features.train` and a testing file `4-features.test`, train your classifier on the datapoints within `4-features.train`, and for each point in `4-features.test`, print the probability that that datapoint's label is 1 from your model. You should assume a uniform distribution over the datapoints, and make sure to include the probability of the datapoint in your calculations. For example, if `4-features.test` has 5 datapoints, a sample output might be:

```
>> python nb_assumptions.py 4-features.train 4-features.test
0.1234567
0.9876543
0.5000000
0.1111111
0.9999999
```

Please record the resulting probabilities from `assumptions/4-features.test`, trained on `assumptions/4-features.train` in `4-result.txt` and include this file in your submission.

6.1 Breaking the assumption

Recall the Naive Nayes independence assumption:

> Each feature is independent of any other feature conditioned on the label

The data in `4-features.train` roughly follows this assumption. For the following section, you will be using the dataset `5-features.{train|test}`, which is exactly identical to `4-features`, except that the last feature is duplicated. Notice that this blatantly disregards the assumption.

Modify your `nb_assumptions.{py|java|c|cpp}` to be able to also handle this new dataset. (Note: If your language is not conducive to handling an arbitrary number of features, you can make the file take in the number as a command line argument)

Compare[2] the outputs of

```
python nb_assumptions.py 4-features.train 4-features.test
python nb_assumptions.py 5-features.train 5-features.test
```

In `assumptions.txt`, **answer the following questions**:

1. How do the probabilities compare between the two datasets?
2. Why is this the case?
3. Could the duplicated feature cause an opposite effect for a different dataset, or will the general effect always be the same? Briefly describe an example or give a short justification.

7 Autolab Submission

Submit a .tgz containing your source code, written assignments, and a file collaboration.txt). You can create that file by running **tar -cvf hw9.tar *.{py|java|c|cpp} *.txt**. **DO NOT** put the above files in a folder and then tar gzip the folder. You must submit this file to the "homework9" link on Autolab.

[2]Comparing the probabilities only makes sense if you account for $P(x)$