UNIVERSITY OF TEHRAN, DEPARTMENT OF NETWORK SCIENCE AND TECHNOLOGIES
FACULTY OF NEW SCIENCE & TECHNOLOGIES

Homework 2
PRML

October 17, 2019

Fall 2019

General Homework Policies

1. Due date of this homework is on Thursday 9 Aban 97 (21 October. 2019), so you need to submit it before the due date[midnight 21 Day] otherwise you wont get any score!

2. Try to budget your time because due dates are hardly changeable, and we will not accept late homework for any reason.

3. You are welcome to collaborate, cooperate, and consult with your classmates provided that you write-up the solutions independently.

4. Dont plagiarize! Write everything in your own words, and properly cite every outside source you use. Taking credit for work as well as ideas that are not your own is plagiarism. Students who plagiarize will not get any score and they will be introduced in the class.

5. Please create reference for all sources(books, papers, websites) which you use.

6. Please create a cover letter for your report which is simply is the Homework#, title of the course, your name, surname, and student number.
7. Email your final file of assignment to sajjadaghapour@ut.ac.ir and ah.havvaei@ut.ac.ir with subject $PRML\_hw\#\_Surname$ which # indicates number of the home work.

Tokenization

Tokenization is a necessary first step in many natural language processing tasks, such as word counting, parsing, spell checking, corpus generation, and statistical analysis of text.
Tokenizer is a compact pure-Python (2 and 3) module for tokenizing Icelandic text. It converts Python text strings to streams of token objects, where each token object is a separate word, punctuation sign, number/amount, date, e-mail, URL/URI, etc. It also segments the token stream into sentences, considering corner cases such as abbreviations and dates in the middle of sentences.

The tokenize() function

To tokenize a text string, call tokenizer.tokenize(text, **options). This function returns a Python generator of token objects. Each token object is a simple namedtuple with three fields: (kind, txt, val).

```
for token in tokenizer.tokenize(mystring):
    kind, txt, val = token
    if kind == tokenizer.TOK.WORD:
        # Do something with word tokens
        pass
    else:
        # Do something else
        pass
```

Alternatively, create a token list from the returned generator:

```
token_list = list(tokenizer.tokenize(mystring))
```

For more tips and information, see the following link: https://pypi.org/project/tokenizer/

Overview

In this assignment you will build a Naive Bayes classifier that can classify movie reviews as either positive or negative. This assignment also asks you to evaluate and analyze your system. The goal is for you to begin to understand the Naive Bayes model, its strengths and weaknesses, how its parameters affect its accuracy and how to use the model to do some exploratory data analysis.

Dataset

Youll be working with the IMDB Large Movie Review Dataset. The dataset includes 25,000 movie reviews for training and 25,000 movie reviews for testing. The root directory of the dataset contains a **train** directory and a **test** directory. Each of these directories has a pos and a neg directory containing positive and negative movie reviews respectively. Make sure that when training your model you use the files in the **train** directory; make sure that when testing your model you use the files in the **test** directory. A copy of the dataset is available on Piazza in the post introducing this homework.

Running and Testing Code Weve partially implemented **nb.py** for you. Fill in the missing code as directed by this assignment. The end of the file contains a runnable main function (run **python nb.py**). The code calls **produce_hw2_results()** which you can continue to implement to produce your results. Feel free to implement whatever other helper functions you like. By default it is set to only use 20 documents: see the **num docs** parameter for the training function. This is to make it easy to very rapidly test code. For all results you report in your solutions, always use the full dataset.

Deliverables and Due Date You should submit a zipped directory named hw2_YOUR Student No. that contains:

- a document containing your responses to all of the questions in this assignment sheet (preferably in PDF format). Make sure to include any and all plots. If you have to write any of the answers by hand, please scan them and include them in your write up.

- any code you wrote for this assignment. This should at least include your completed nb.py file and may also include a completed **produce_hw2_results()** function.

- an estimate of the number of hours this assignment took you to complete (and any additional feedback).

Your work must be submitted via moodle no later than midnight on Thursday, October 31th. Our courses collaboration policy is specified on the website.

1 Tokenization, Bag-of-words and Counting

Before building the model, youll need to get the text into a representation that the model can handle. Recall from lecture that the Naive Bayes model makes use of a *bag-of-words representation*. Naive Bayes is order-independent in that it doesnt care about the order of the words in the documents it classifies; it only keeps track of the number of each word type it encounters.

1. **(5 pts)** Implement the **tokenize doc** function. This function should take a document (as a string), split the document into tokens (for now just split on whitespace) and return a mapping (dictionary) of each token to the number of times it appears in the document. Every token in the document should be lower-cased. Test your function afterwards by uncommenting the relevant lines in the **produce_hw2_results** function and running **python nb.py**.

2. **(5 pts)** Implement the **update_model** function. Before you start, make sure to read the function comments so you know what to update. Also review the **NaiveBayes** class variables to get a sense of which statistics are important to keep track of. Run the **train_model** function with your new tokenization code. What is the size of the vocabulary used in the training documents? Youll need to provide the path to the dataset you downloaded to run the code.

3. **(2.5 pts)** Lets begin to explore the count statistics stored by the **update_model** function. Implement the **top_n** function. This function takes a sentiment class and a number, **n**, a returns the top **n** words sorted by the number of times they appear in documents of that class. What are the top 10 most common words in the positive class? What are the top 10 most common words in the negative class?

4. **(2.5 pts)** Will the top 10 words of the positive/negative classes help discriminate between the two classes? Do you imagine that processing other English text will result in a similar phenomenon?

2 Word Probabilities and Pseudocounts

The Naive Bayes model assumes that all features are conditionally independent given the class label. For our purposes, this means that the probability of seeing a particular word in a document with class label y is independent of the rest of the words in that document.

1. **(5 pts)** Implement the **p_word_given_label** function. This function calculates P($w|y$) (i.e., the probability of seeing word w in a document given the label of that document is y).

2. **(5pts)** Use your function to compute the probability of seeing the word "fantastic" given each sentiment label. Repeat the computation for the word "boring." Which word has a higher probability given the positive class? Which word has a higher probability given the negative class? Is this what you would expect?

3. **(2.5 pts)** What happens if you try to compute the probability of a word that exists in the positive training data but not in the negative training data (and vice versa)? Explain what is going wrong.

4. **(5pts)** We can address this issue with *psuedocounts*. A psuedocount is a fixed amount added to the count of each word stored in our model. Psuedocounts are used to help smooth calculations involving words for which there is little data. Implement **p_word_given_label_and_psuedocount**.

3 Prior and Likelihood

As noted before, the Naive Bayes model assumes that all words in a document are independent of one another given the documents label. Because of this we can write the *likelihood* of a document as:

$$P(w_{d_1}, \cdots, w_{d_n}|y_d) = \prod_{i=1}^{n} P(w_{d_i}|y_i) \tag{3.1}$$

where w_{d_i} is the i^{th} word in document d and y_d is the label of document d.
However, if a document has a lot of words, the likelihood will become extremely small and well encounter numerical underflow. Underflow is a common problem when dealing with probabilistic models; if you are unfamiliar with it, you can get a brief overview on Wikipedia: https://en.wikipedia.org/wiki/Arithmetic_underflow. To deal with underflow, a common transformation is to work in log-space.

1. **(5 pts)** Derive the log of the likelihood function above.

2. **(5 pts)** Implement the **log_likelihood** function. Hint: it should make calls to the **p_word_given_label_and_psuedocount** function.

3. **(2.5)** (2.5 pts) Implement the **log_prior** function. This function takes a class label and returns the log of the fraction of the training documents that are of that label.

4 Normalization and the Decision Rule

Naive Bayes is a model that tells us how to compute the posterior probability of a document being of some label (i.e., $P(y_d|w_d)$). Specifically, we do so using bayes rule:

$$P(y_d|w_d) = \frac{P(y_d)P(w_d|y_d)}{p(w_d)} \tag{4.1}$$

In the previous section you implemented functions to compute both the log prior ($log[P(y_d)]$) and the log likelihood ($log[P(w_d|y_d)]$). Now, all your missing is the normalizer ($P(w_d)$).

1. **(5 pts)** Derive the normalizer.
2. **(5 pts)** Derive the log of the posterior probability by taking the log of the equation above.
3. **(5 pts)** One way to classify a document is to compute the unnormalized log posterior for both labels and take the argmax (i.e., the label that yields the higher unnormalized log posterior). The unnormalized log posterior is the sum of the log prior and the log likelihood of the document. Why dont we need to compute the log normalizer here?
4. **(2.5 pts)** Implement the **unnormalized_log posterior** function.
5. **(5 pts)** Implement the **classify** function. The classify function should use the unnormalized log posteriors but should not compute the normalizer.

5 Evaluation

After training our model and implementing the classify function wed like to evaluate its accuracy.

1. **(10 pts)** Implement the **evaluate_classifier_accuracy** function. This function should classify all of the instances in the test set and report the fraction of instances that are classified correctly. Report your classifiers accuracy (with psuedocount parameter 1.0).
2. **(5 pts)** Experiment with the effect of varying the psuedocount parameter on classifier accuracy. Plot classifier accuracy as a function of the psuedocount parameter. We have provided you with some sample code (the function **plot_psuedocount_vs_accuracy**) to help get you started with plotting. You may want/need to modify this function.
3. **(5 pts)** Find a review that your classifier got wrong. Why do you think your system misclassified this example? What improvements could you make that may help your system classify this example correctly?

6 Exploratory Analysis

Our trained model can be queried to do exploratory data analysis. We saw that the top 10 most common words for each class were not very discriminative. Often times, a more descriminative statistic is a words likelihood ratio. A words likelihood ratio is defined as

$$LR(w) = \frac{P(w|y = pos)}{p(w| = neg)} \tag{6.1}$$

A word with $LR = 5$ is five times more likely to appear in a positive review than it is in a negative review; a word with $LR = 0.33$ is one third as likely to appear in a positive review than a negative review.

1. **(2.5 pts)** What is the range of the LR function?

2. **(2.5 pts)** Implement the **likelihod_ratio** function. This function takes a word and computes the likelihood ratio as defined above.

3. **(2.5 pts)** What are LR("fantastic") and LR("boring")? Compare these to the likelihood ratio of some of the words in the top 10 lists generated above. For example, compare them to LR("the") and LR("to").

4. **(5 pts)**Explain how the word LRs are related to the Naive Bayes classifier model. If a word has LR=1, does that mean the word is or is not important for the NB classifier? If a word has LR very far from 1 (for example, LR=0.01, or LR=100) does that mean the word is or is not important for the classifier? What does an LR=0.01 word indicate, as compared to a LR=100 word, for the operation of the classifier? Explain.

7 Bonus

Often times we care about multi-class classification rather than binary classification.

1. **(6 pts)** How would the count statistics that we are storing change if the model were modified to support multi-class classification? classify all of the instances in the test set and report the fraction of instances that are classified correctly. Report your classifiers accuracy (with psuedocount parameter 1.0).

2. **(2 pts)** How would the normalizer change?.

3. **(2 pts)** What would be the new decision rule (i.e., how would the classify function change)?