

**Text Classification (Naïve Bayes)**

**Naive Bayes classifier.** Write a Python program that implements the Naive Bayestext classifier, as discussed in class. To avoid zero counts, make sure you also implement the add-one smoothing.

Evaluate your implementation on the provided **deception dataset**.

The dataset consists of 196 files, all of them representing statements about a best friend, half of which are deceptive and half of which are truthful. The ground truth (i.e., label) for each statement is encoded in the filename; for instance, the statement stored in the file lie13 is deceptive.

For evaluation, use the leave-one-out strategy, meaning that you train your Naive Bayes classifier on 195 files, and test on the remaining one file. Repeat this process 196 times.

Programming guidelines: Write a program called *naivebayes.py* that trains and tests a Naive Bayes classification algorithm. The program will receive one argument on the command line, consisting of the name of a folder containing all the data files.

Include the following functions in *naivebayes.py*:

a. Function that trains a Naive Bayes classifier: Name: *trainNaiveBayes*; input: the list

of files to be used for training; output: data structure with class probabilities; output: data structure with word conditional probabilities; output: any other parameters required (e.g., vocabulary size)

Given a set of training files, this function will: preprocess the content of the files provided as input, i.e., apply *tokenizeText*. In the basic implementation, do not remove stopwords, do not use stemming. Calculate all the counts required by the Naive Bayes classifier

b. Function that predicts the class (truth or lie) of a previously unseen document. Name:

*testNaiveBayes*; input: the file to be used for test; output: predicted class (truth or lie)

The main program should perform the following sequence of steps:

I. Open the folder containing the data files, included in the folder provided as an argument on the command line (e.g., bestfriend.deception.training/), and read the list of files from this folder. Repeat 196 times:

II. Select one file as test, and the remaining as training.

III. Apply the *trainNaiveBayes* followed by the *testNaiveBayes* functions.

IV. Determine if the class assigned by the testNaiveBayes function is correct. The *naivebayes.py* program should be run using a command like this:

*% python naivebayes.py bestfriend.deception.training/*

It should produce a list consisting of pairs of file names, along with the class predicted by your implementation. E.g.:

lie1.txt

lie lie2.txt

true true1.txt

true true2.txt

true etc.

It should also output the accuracy of your classifier, calculated as the total number of files for which the class predicted by your implementation coincides with the correct class, divided by the total number of files.

Write-up guidelines:

Create a file called answers.txt.

Include in answers.txt the following information:

1. Accuracy of your Naive Bayes classifier, as described above
2. Accuracy of your Naive Bayes classifier, when you also remove the stopwords
3. Accuracy of your Naive Bayes classifier, when you also stem the words

1. Accuracy of your Naive Bayes classifier, when you also remove the stopwords and stem the words

Using the implementation that does not remove stopwords and does not stem words, list the top 10 words that have the highest conditional probability (i.e., P(w|c)) in each of the two classes considered (truth, lie). Under each class, list the words in reversed order of their conditional probability.

What to submit:

You should submit a zip file containing your source code (.py), an (optional) report and answers.txt. Name your zip file as StudentNumber\_Lab4.zip and upload to LMS.