Programming Assignment 4 – Basic Probability, Computing and Statistics 2016

Fall 2016, Master of Logic, University of Amsterdam

Instructors: Philip Schulz, Christian Schaffner and Bas Cornellisen

Submission deadline: Wednesday, November 30th, 8 p.m.

Note: if the assignment is unclear to you or if you get stuck, do not hesitate to ask in the forum.

1 Assignment

This week we will implement a Naïve Bayes (NB) classifier for text data. If you do not remember what a NB classifier is, check here. We will call the variable whose value we try to infer (Y in the script) the class, label or category (all three are equivalent). The data point x_i are generally called **features**. Here, we take our features to be words. We will first train the classifier using MLE and then make predictions based on our estimates.

1.1 Data

We will use the famous 20 newsgroups data set for this exercise. As the name suggest, there are 20 groups of text between which we have to distinguish. This implies a baseline performance achieved by randomly choosing a group of 5%. The data are emails within newsgroups collected in the 1990's. For a start, we will simply use all strings separated by whitespace as features for our NB model. We call these strings words.

1.2 Smoothing

Many words will only occur in one class c. When we see a text from that class during prediction time, the other classes will not have probabilities for as many words as c. There are two ways to go about this:

1. Simply ignore the words that have 0 probability under a class. This will lead to many classes having fewer terms in the NB product than c. Thus, these classes will have a higher posterior.

2. Do model all words for all classes. Unfortunately, most unseen texts (the ones we are dealing with at prediction time) contain for each class at least one word that has 0 probability. In effect, the posterior probability for all classes will be 0. Notice that this means the posterior is not even defined as it does not sum up to 1.

As you can see, both ways will give us the wrong results. There is a cleverer way, however. We remember all words that we have encountered during training. These define our vocabulary. During prediction we will ignore all words not in the vocabulary. Moreover, we perform smoothing. This means that, before doing MLE, we add a constant count for *all* words in the vocabulary to the training counts of every class. This has the effect that all vocabulary words will have positive probability under all classes.

1.3 Logprobs

Last week we mentioned that in probabilistic calculations, a product of probabilities can become so small that it cannot be represented by your computer anymore. This is exactly the case with Naïve Bayes. When you classify a document, you have to compute a potentially large product of probabilities. To avoid turning them to 0 (and to make computation more efficient), we will work with log-probs instead. Make sure to transform your MLEs to log-probs before doing any predictions.

1.4 Your Task

We have implemented a commandline interface and a skeleton of the Naive Bayes class for you. Both can be found here. Please do not manipulate `naive_Bayes_classifier.py`. Only work on `Naive_Bayes.py`. Feel free to add any methods and data structures that you deem necessary.

Challenge yourself On the development set, which we provide together with the code, you should achieve an accuracy of 80.95% if the smoothing constant is set to 1. Can you do better than that? Try different smoothing constants or try to change the words (e.g. by lowercasing them) or try to include more features (characters, for example). The possibilities are limitless!

1.5 Running the Code

You will have to supply commandline arguments to the script this time. This can be done in Pycharm. Right-click on the run symbol and select *Add parameters*. In the field *script parameters* you then have to enter the following for `naive_Bayes_classifier.py`

```
--training-corpus-dir <Path to 20news-18828>
   --test-set-directory <Path to dev-set>
```

(Note: the line break was inserted by LaTeXand is not needed in Pycharm.) The output will be a file called `predictions.txt`. You can check your performance using the accuracy checker. For that, you will have to pass argument to `accuracy_checker.py`, as well. These are simply

```
<Path to predictions.txt> <Path to dev_keys.txt>
```

2 Grading

You will be graded on a test set that we will make available during peer review. The test set is structured like the dev set but contains different files.

2 points	The code produces predictions that are in the correct format and assign labels that it has been trained on (instead of arbitrarily named labels)
2 point	The test-set (not dev-set(!)) accuracy is at least 40%.
2 point	The test-set (not dev-set(!)) accuracy is at least 60%.
2 point	The test-set (not dev-set(!)) accuracy is at least 70%.
1 point	The test-set (not dev-set(!)) accuracy is at least 80%.
1 point	The test-set (not dev-set(!)) accuracy is higher than 80%.