Arvind Ramanathan
aramanat@utk.edu

# Naive Bayes classification of news articles

## Background

In this assignment, we are going to use the Naive Bayes algorithm as a means to automatically classify news reports. In particular, we will build on the material that we presented in class and test the classifier's performance using different settings.

This will be a tutorial like walk-through of setting up Apache Spark, understanding the use of PySpark bindings and eventually to use the MLlib functions to set up, develop, and test a simple classification application.

Please don't go out searching for existing code. As such, in this assignment, we will be making use of existing libraries within Spark (i.e., MLlib) and other open source code to pre-process/post-process our analysis. If you are stuck, please try to email the list. Everyone will face similar issues in implementing the assignment and my suggestion is to collaborate and help figure out the implementation.

### What to expect? What to hand in?

For this assignment, you can collaborate, but your reports and code must be your own. The total points on this assignment is 100, it is worth 15% of your overall grade.

Your write up should consist of the following subsections: (1) Collaborators, (2) Approach, and (3) Results. I expect that you will typeset your report. If you do not typeset your report, I will deduct 10 points from the final grade.

### Deadlines

Deadlines are firm. You have up to three (3) "dog days" total (for the entire semester). Please email me if you need to use your dog days. Please submit your assignment by **11.59 PM on Feb 16, 2018**.

### Programming

Please use Python for your programming environment.

Email me and Yongli Zhu (yzhu16@vols.utk.edu) with the following attachment [lastname]-HW1-submit.tgz on or before Feb 16, 2018. On the subject line, please include the following tag- [COSC-526: HW1].

## Dataset and pre-processing steps [25 points]

We will use the Reuter's news articles dataset as our input data. The Reuter's dataset (Reuters-21578) is a good example of how news articles are usually available on your typical RSS feed. The data was originally collected by Reuters and is already been labeled. You can download it here:

https://www.dropbox.com/s/jlgd0o8su4spo8k/reuters21578.tar.gz?dl=0

As a dataset that is not fully pre-processed, you will have to first extract the data that is relevant to your assignment. The datasets contains 22 .sgm documents and each files contains approximately 1000 papers of various topics. In this assignment we will only focus on the following topics: "money, fx, crude, grain, trade, interest, wheat, ship, corn, oil, dlr, gas, oilseed, supply, sugar, gnp, coffee, veg, gold, soybean, bop, livestock, cpi."

You will pre-process the documents to construct your training and testing datasets. First, parse the XML documents using the Python's built-in SGML library and extract only the topics and body of the article. Next, construct a list that allows you to organize your data as `docs = [(topic, body)]`. Once you process all of the data, filter out the text that belongs to the aforementioned categories only [10 points].

Next, you will have to tokenize your documents to keep only the meaningful content relevant to the classification task. Usually this can be challenging. One aspect that is particularly annoying is the fact that you will have to worry about words ending in 'ing' or other modifiers that may artificially increase the size of your vocabulary. So, you need a way to make sure that you are counting the right occurrences and make sure you get rid of the annoying modifiers, stop words, etc. For this, we will rely on using the natural language processing toolkit (NLTK) from Stanford. You can use the PorterStemmer class to get the right tokens [10 points].

The final step will be to create your dataset that you can use for training/testing. Write out your file and name it as training_test_data.txt. From this file, simply print the first 10 (topic, body) tuples into a text file [5 points].

## Building a representation that transforms text for learning [25 points]

Now that you created your dataset, you will want to create a representation that allows for you to build a Naive Bayes classifier. For this purpose, you will transform your tokens using a term-frequency inverse document frequency (TF-IDF) representation. You can learn more about TF-IDF here: https://en.wikipedia.org/wiki/Tf-idf.

Your first task will be to implement a TF-IDF function without Spark functionality. For this purpose you can depend on NLTK or even `scikit.learn` library to process the documents and obtain a TD-IDF transform.

As you can imagine, PySpark also provides easy functions to obtain TF-IDF transformation. Let's say you read the text file from the previous step into the variable called `data`, then you can transform data using the `HashingTF` class from Spark's MLlib library. A sample listing will look like this:

```
from pyspark.mllib.feature import HashingTF, IDF
tf = HashingTF().transform(data.map(lambda doc: doc[0]["text"],
    preservesPartitioning=True))
idf = IDF().fit(tf)
tdidf = idf.transform(tf)
```

You will report the time it takes for obtaining the TF-IDF transform with/without Spark [20 points]. How do the TF-IDF from Spark's internal library and scikit.learn/NLTK compare? [5 points].

After this step, you will essentially combine the labels with the TF-IDF transformed data to create a `LabeledPoint`. A labeled point essentially provides you with access to a document body that is explicitly labeled with the topic. A snippet shown below can serve as a starting point for you.

```
dataset = labels.zip(tfidf).map(lambda x: LabeledPoint(x[0], x[1]))
```

## Training Naive Bayes [35 points]

After you create your dataset, you will create a training/testing/validation split of the data. First, reserve 10% of the data as your validation set. Then use three splits on the remaining 90% of the data: (1) 50/40, (2) 60/30, and (3) 70/20. Your training/testing data should be implemented using `randomSplit` function. You will invoke the `NaiveBayes.train()` function to build your model. Train the model (for each set of splits) 10 times and report the average accuracy on the remaining testing dataset for each split [20 points].

What are the trends you observe as you increase the amount of training data? [15 points]

You will do well by saving the models that you trained as part of your training process. We will use these models to test our Naive Bayes classifier on the validation dataset.

## Testing Naive Bayes [15 points]

Now, use the validation dataset and report the accuracy of the classifier on each of the models that you built. This means, for the final testing, you will end up having 30 models! Report the best classifier performance on the validation dataset.