MCIS6273 Data Mining (Prof. Maull) / Fall 2020 / HW3

This assignment is worth up to 15 POINTS to your grade total if you complete it on time.

Points Possible	Due Date	Time Commitment (estimated)
15	Wednesday, November 18 @ Midnight	*up to* 8 hours

- **GRADING:** Grading will be aligned with the completeness of the objectives.
- **INDEPENDENT WORK:** Copying, cheating, plagiarism and academic dishonesty *are not tolerated* by University or course policy. Please see the syllabus for the full departmental and University statement on the academic code of honor.

OBJECTIVES

- Perform Bayesian text classification

WHAT TO TURN IN

You are being encouraged to turn the assignment in using the provided Jupyter Notebook. To do so, make a directory in your Lab environment called `homework/hwN`. Put all of your files in that directory. Then zip that directory, rename it with your name as the first part of the filename (e.g. `maull_hwN_files.zip`), then download it to your local machine, then upload the `.zip` to Blackboard.

If you do not know how to do this, please ask, or visit one of the many tutorials out there on the basics of using zip in Linux.

If you choose not to use the provided notebook, you will still need to turn in a `.ipynb` Jupyter Notebook and corresponding files according to the instructions in this homework.

ASSIGNMENT TASKS

(100%) Perform Bayesian text classification

Text classification is an important application area of machine learning. Indeed, the early advances in the field were in text and image processing. We are now beneficiaries of the libraries that provide us the foundation for a variety of techniques to do sophisticated text analytics and process withouut much effort.

With text classification, one goal we might like to accomplish is determine the original of particular text. What once used to be the arena of computational linguists and computer scientists, is now growing in computational digital humanities, but is not without issues. In this assignment we are going to use Bayesian techniques to process a corpus, or body of text, with the expressed goal of classifying it. In fact, we're going to take multiple texts and generate a classifier that (with some work), will be able to distinguish between multiple authors.

Laying out the intuition of the technique, let's abstractly think about the problem at hand. An author usually writes in a particular style, uses specific words, chooses particular topics, and as such, these things contribute to a "profile" (loosely speaking) of the author's works. More concretely, the probability that Albert Einstein used a word like "thou" is much lower than the probability that Shakespeare would use the same word. Ultimately with enough examples of the writings of a particular author, the easier it would be to establish the probabilities of certain words, phrases and even punctuation usage. While we are going to choose to classify authors, the same technique could be used to classify content themes, for example, classifying texts about *philosophy* versus texts about *turtles*.

Bayesian techniques are a mainstain, and the ease with which Bayes classifiers can be trained make it a technique that can be fast to implement and get results that are often very accurate.

In the interest of time and resouces, we're going to develop a simple Bayesian text classifier to distinguish between the writings of three philosophers – Aristotle, Plato and David Hume. Under ordinary circumstances we would like to have as many documents from each of these authors as possible – instead, we will train from 3 documents each and test against 3 more. Again, if we were going to develop a much more robust classifier we would likely choose a much larger number of documents if they were possible. Luckily, there are enough writings to have enough to work with in this abbreviated exploration.

At the heart of document classification is the *model* for document features. One popular model is the TF-IDF or Term Frequency Inverse Document Frequency. The intuition behind analyzing words in documents hinges on the following:

- terms that are frequent *in documents* are given higher importance than those that are infrequent,
- terms that are frequent *across* documents are not considered as important;

that is *common* words across an entire corpus are *discounted* while those that are *common* within documents are *boosted*. This is an effective way to differentiate since the intuition that the things that make your writing unique are amplified, while those that are not differentiators will count less.

To realize the TF-IDF, we will need to break apart the two components TF (or **term frequency**) and IDF (**inverse document frequency**) and then conjoin them.

Term frequency (TF) is a simple concept and is exactly as it says: the *counts* of terms in a document. So for a term (word) t and document d, the TF is just the number of occurences of t in d,

$$\text{tf}(t, d) = \big|t \in d\big|$$

Inverse document frequency (IDF) provides a way to determine if a terms is rare or common given *all* documents D, and is logarithmically scaled so rare terms avoid completely disappearing. Thus,

$$\text{idf}(t, D) = \frac{\big|D\big|}{1 + \big|\{t \in d \big| d \in D\}\big|}$$

TF-IDF is thus: for a set of documents (corpus) D and document $d \in D$ and terms $t \in d$,

$$\text{tfidf}(t, d, D) = \text{tf}(t, d, D) \times \text{idf}(t, D)$$

Luckily, `sklearn` implements TF-IDF for us in the `sklearn.feature_extraction.text.TfidfVectorizer` class. The underlying implementation uses the words as the feature matrix where the TF-IDF is computed over every document input to the `vectorizer.fit_transform()` method.

Now that we've implemented to the primary machinery of the method, let's bring back Bayesian. Recall the Bayesian method:

$$\Pr(C\big|w_1, \ldots, w_n) = \Pr(C)\prod_i^n \Pr(w_i\big|C)$$

where C is the document class (Plato or class `A`, Hume or class `B` and Aristotle or class `C`) and w_i the words in the document. Concretely, a document D_i has some probability P_i based on the occurrence of the words w_i in that document, and that a classifier will decide the class $\hat{C}$ of document D_i by computing

$$\hat{C} = \text{argmax}_C \Pr(C)\prod_i^n \Pr(w_i\big|C)$$

by training the classifier on some labeled data. Once trained the classifier can be tested and then used on unlabelled data to classify the author. While this exercise is decidely oversimplified (we'd not really be all that interested in classifying the works of only 3 authors), you can extend this to other domains where perhaps you're not classifying authors, but styles, topics or document complexity.

Completing the assignment will require you use the provided notebook and corresponding data files. This notebook can be found in `example_notebook.ipynb`. Study it closely.

§ Using the notebook provided and corresponding files, you are to write a Python function to load the training data. Use the dictionary map in the variable `training_map`. Your function will take the files (in the order they appear in `training_map`) and pass the data into the `TfidfVectorizer` vectorizer. You will need to set the parameter to the constructor to `input='file'` and the `stop_words` to `'english'` (e.g. initialize the vectorizer to `TfidfVectorizer(input='file', stop_words='english')`.

- **You will just need to show the new function and the initialization of the vectorizer in this step.** This will be one or two cells at most.
- You will use `fit_transform()` with the parameter being a list of the training files.

§ Now that you have a vectorizer which effectively builds the data structure to hold the TF-IDF of all the words which appear for each document, you can move to the training phase for the Bayesian classifier. Look in the sample notebook for guidance. You will take as input the vectorizer output (the documents vectorized by TF-IDF) and the corresponding classes (in the order they appear in the original dictionary map) and pass that into the `MultinomialNB.fit()` method.

- **Show the initialization of your `MultinomialNB()` classifier and the application of the `fit()` method.**

§ Now that you have a vectorizer and a classifier trained, let's classify some documents. The vectorizer you created in the first step has a `transform` method, which you can use to vectorize a new document and pass as test input to the classifier. Once you have the vectorized version of the document, you can use the classifier's `MultinomialNB.predict(test_data)` method, which will return the class label according to the classifier.

In your notebook do the following:

1. **Write a function to take as input a vectorized document and trained classifier and return the predicted label for the document.** See the sample notebook for guidance.
2. **Test on the files in the `data/philosopher_name/test` folders and show the output of your test.** You can wrap your function from the previous step in a loop to run through all data in the folder. This will be short enough to be coded in a single Jupyter cell.

To get the full points, please show your work.

§ You have now built your first document classifier! Now answer the following questions:

1. How many of the documents did your classifier correctly classify?

§ The classifier `predict` method only returns the label, but you can get the probabilities assigned to all classes using `predict_proba()`. Please take a look at the example notebook to see how that is done.

Answer the following questions inside your notebook:

1. Make an observation about the class probabilities. What did you notice?
2. Provide some commentary on how the probabilities might be improved (you can provide you answer as a thought exercise or if you have time, provide some example code).