MCIS6273 Data Mining (Prof. Maull) / Fall 2021 / HW3b

This assignment is worth up to 10 POINTS to your grade total if you complete it on time.

Points Possible	Due Date	Time Commitment (estimated)
10	Wednesday, Devember 1 @ 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 and modules that provide us the foundation for a variety of techniques to do sophisticated text analytics and process withuout much effort.

With text classification, one goal we might like to accomplish is determine the origin 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 topics.

As a graduate student, you are fully aware of the extent of academic research represented by the multitude of disciplines in our university community. You are likely well aware of the many thousands of academic journals that contain the intellectual products of the research in those disciplines. We are going to build a Bayesian classifier that will be trained on the text from abstracts of academic papers, and be able to classify unseen (unlabeled) abstracts into their corresponding disciplines.

For this assignment we are going to keep it simple, mostly just to get you started on the using technique so that you might extend it and learn ways to improve it in the future.

Laying out the intuition of the technique, let's abstractly think about the problem at hand. An academic discipline usually contains a large domain-specific vocabulary that make it unique relative to other disciplines. Think of the words in the vocabulary as a "profile" (loosely speaking) of the discipline. More concretely, the probability that computer science papers used a word like "algorithm" is much higher than the probability of "algorithm" being

used in an education paper. Ultimately with enough examples of the writings of a particular discipline, 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 papers, the same technique could be used to classify authors, for example, classifying texts written by *Don Knuth* versus texts authored by *Noam Chomsky*.

Bayesian techniques are a mainstain in text classification of all kinds, 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 five disciplines: *sociology*, *education*, *physics*, *computer science* and *economics*.

Under ordinary circumstances we would like to have as many documents from each of these disciplines as possible, and as you might know, obtaining full text documents is often difficult or requires extraction of raw text from PDF documents (that are often only obtained under publisher license or may require payment). For more information about how extract text from PDFs in Python, please see the pdfminer.six module as it contains a number of wonderful functionalities to get the text portion of a PDF so it can be processed by more common text and string processing tools.

Instead of full text PDF documents, we are lucky in academia to have *abstracts* which summarize the paper for the reader. These usually paragraph-long text are often enough to get a good idea about the thrust of the paper, and as we have learned in this class *large amounts* of data are often necessary for any meaningful test of an algorithm.

In class we talked about test sets, training sets and the permutations we might conduct to get a mix of test/train sets to build supervised learning algorithms. You are being provided with a small set of testing and training documents for each discipline, and will use these documents and the provided Jupyter Notebook that step through the process of building the naïve Bayes classifier with Sci-Kit Learn's algorithms.

We would like to have as many documents as possible – but in the spirit of time, we will instead use tens of abstracts from each discipline and those will act as the training corpus. In a more robust classifier training, we will want hundreds or even thousands of abstracts, and as you will see this will impact the results as well as bias the classifier.

Document Processing: A Very Short Primer

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,

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

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,

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

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:

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

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

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

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 5 disciplines over a narrow number of test instances), you can extend this to other domains where perhaps you're not classifying topics, but styles, tone or even 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, execute the notebook to load the training data.

You will do this by uncommenting the cell that sets the `file_list` for loading the corpori. Once you execute this cell the classifier will be trained and you can then test it.

- **You will just need to show the uncommented code and initialization of the classifier in this step.**
- Open the files in `data/train` and explore their contents. You will notice these are just large numbers of words that have come directly from the abstracts. Later we will show how to build your own.

§ Now that you have a classifier, the real work is to be done – testing. I have provided a set of test documents in the `data/test/` folder.

Write a cell that loads a the list of documents in the `data/test` folder and passes that list to the function `vectorizer.transform()` (this will be the same `vectorizer` in the prior cell). See the example notebook for more extensive information on how you might do this.

§ Now that you have a vectorizer and a classifier trained and tested you might notice that the classifier is not that great just yet on the limited data.

You might have noticed that sociology, computer science and eduction are seemingly similar. Let's try to do better by expanding the computer science corpus.

You will see a file in `data` called `seed_doi_compsci`. In it are over 100 DOIs to a more extensive computer science corpus. A DOI or Digital Object Identifier, is permanent identified for an academic paper (or technically any digital asset). It allows any machine to lookup the *metadata* of the DOI and then also resolve the actual object, whether it is a PDF, dataset, software or whatever it resolves to.

There is a service called Semantic Scholar which provides an academic service for research papers and their networked connections to other relevant papers and authors, and a host of relevant metadata through an API that provides abstracts and other metadata from a single service, which would otherwise be difficult to obtain.

Another service crossref provides similar API support for publication metadata and the two services complement each other.

We're going to now have some fun with APIs and extract the abstracts for all of those DOIs in the seed file, then retrain the classifier.

You can use the API as provided in the semantic scholar documentation and in particular you will use the lookup functions provided by the API to lookup by DOI. Once you have the JSON object back, you can grab the abstract from that.

You are also free to install the Semantic Scholar Python package from PyPi which hides most of the processing and API calls and just provides the essential functions to process the return object in a natural programmatic way. I have used the package and it is very good.

NOTE: *Semantic Scholar rate limits your calls to 100 per 5 minutes or 1 every 3 seconds / 20 per minute. You will want to use the Python* `time.sleep(n_sec)` *function to avoid being shut down for API timeout / cooloff. Please be nice and usually I set the sleep to 5 seconds, just to be safe.*

In your notebook do the following:

1. **Write a function to take as input DOIs in `seed_doi_compsci` and produces a new file in the `data/train` folder called `compsci_extended.txt`.** This file will just contain the contentation of all the abstracts. You will use 75 of the abstracts as the test set and the remaining as the training set.
2. **Retrain the classifier with the extended corpus instead of the original one.**
3. Test the new classifier on the remaining DOIs which you did not train. Please show the outcome of the test.
4. How well did the new classifier do on the test data? In your reporting of this number, list the percent of correctly classified documents (e.g. a perfect classifer will be 1.00 or 100%, and a poor classifer might be 0.333 or 33.3%).

§ The classifier `predict` method only returns the label, but you can get the probabilities assigned to all classes using `predict_proba()`.

Answer the following questions inside your notebook:

1. Make an observation about the class probabilities. What did you notice?
2. Provide some commentary as a thought exercise or if you have time, provide some example code).

§ A bonus assignment that extends this analysis will be provided for you to attempt.