MCIS6273 Data Mining (Prof. Maull) / Fall 2024 / HW1

Points Possible	Due Date	Time Commitment (estimated)
40	Monday December 9 @ Midnight	*up to* 20 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 basic supervised learning Naive Bayes classification.
- Learn about machine learning ethics.

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/hw0`. Put all of your files in that directory. Then zip or tar that directory, rename it with your name as the first part of the filename (e.g. `maull_hw0_files.zip`, `maull_hw0_files.tar.gz`), 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

(70%) Perform basic supervised learning Naive Bayes classification.

In the last homework, you learned how to do unsupervised learning with K-Means, which does not need labeled data to work.

In this part, you will get your feet wet with unsupervised learning using Naive Bayes classification.

In some of the lecture notes, you learned that Naive Bayes can be used to use prior probabilities of know (and unknown) data to determine how well a hypothesis fit data. We thus characterize Bayes like this:

$$\Pr(H|D) = \frac{\Pr(D|H)\Pr(H)}{\Pr(D)}$$

We know that $\Pr(D)$ is a constant and can be dropped in our calculations without loss of generality.

Given this, there are several ways to perform Naive Bayes, and we will be using it to do *classification tasks*. A classification task requires that the target classes of training data are known *a priori*, and we are given *test data* without classes and allow the classifier to assign the class of test data once training is completed.

A common area of classification is document classification where given a a set of training documents $D_T = \{d_{t_1}, \dots, d_{t_n}\}$ and their classes $C_{D_T} = \{c_{d_{t_1}}, \dots, c_{d_{t_n}}\}$, we have a classifier trained on (D_T, C_T). The goal of the classifier is to learn the features of each document and their classes and Naive Bayes provides a way to do this, so that given test documents T (with unknown classes), the performance of the classifier in assigning the correct class is as close to 1 as possible.

Document processing and feature extraction is done, often with word frequency analysis using a technique called TF-IDF, term frequency-inverse document frequency. The *term frequency* uses the frequency of a word in a document to provide a *probability* of that word relative to all other words in a document – think of it as as weighting of the relative importance of the word (i.e. term).

The *document frequency* examines the frequency of a word over multiple documents and essentially determines how frequently a word occurs in a corpus or collection of documents.

We intuit that high frequency words over large corpora do not add anything to our seperation or distinction of the documents. For example, the use of the word “the” is quite high in the English language, and so it does not provide much discriminating power when trying to understand the difference between two documents, but the word would be high frequency in nearly all documents. The word “simulacra” on the other hand, is low frequency and thus could provide discriminating power in assigning the weight of its contribution to a classifier. This would especially be true, if it was used more frequently in one document versus all others. Thus, the IDF component of TF-IDF uses the *inverse* frequency (over all documents) to give higher weight to lower frequency (unique) words, adding to their discriminatory power.

It is observed that combining these two concepts is quite powerful in developing a discriminatory mechanism for document similarity, so that given a large corpus, we can use classifiers trained on TF-IDF to classify new, unseen documents. You will note, that this is precisely how email SPAM filters have worked in classifying suspicious emails. We will see in this part, that it can be useful for much more.

To summarize:

- 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;

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 combine 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 ratio of the number of occurences of t in d to the number of terms in d,

$$\mathrm{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

where f_{t_d} is the number of occurrences of t in d.

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,

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

where N is the number of documents in the corpus.

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

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

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. Let’s recall the Bayesian method and assume our data D are all words w in the corpus and hypothesis H, the classes C, then :

$$\Pr(C|w_1, \dots, w_n) = \Pr(C) \prod_{i}^{n} \Pr(w_i|C)$$

where C is the document class (Author A or class `A`, Author B or class `B` and Author C 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 a few authors), you can extend this to other domains where perhaps you're not classifying authors, but styles, topics or document complexity.

This foundational explanation can be use to understand the explanations in ScikitLearn.

In this assignment we will use the MultinomialNB classifier in ScikitLearn.

§ Task: Enhance the notebook for further use.

I have provide a notebook which you will complete. The notebook is on Github here:

- Example notebook to complete

The notebook includes a function `generate_gutenberg_dataset` which takes the ID of a Gutenbeg text and returns a document with all of the text for those IDs in one document.

You will be using this to build a corpus for 4 authors:

- Herodotus
- Lewis Carroll
- F. Scott Fitzgerald, and
- T. Smollett.

You task is to insert a **single cell** which creates four lists with the following names:

- `herodotus`, `lewis_carroll`, `f_s_fitzgerald` and `t_smollett`

Your code cell will look like this:

```
herodotus = [ 000, 000, 000 ] # put the IDs in the list
lewis_carroll = [ ... ]
f_s_fitzgerald = [ ... ]
t_smollett = [ ... ]
```

but instead of empty lists, you will place the Gutenberd IDs of the following works into the lists.

- `herodotus`:
  - *The History of Herodotus — Volume 1 by Herodotus*
  - *An Account of Egypt by Herodotus*
  - *The History of Herodotus — Volume 2 by Herodotus*
- `lewis_carroll`:
  - *Alice's Adventures in Wonderland by Lewis Carroll*
  - *Through the Looking-Glass by Lewis Carroll*
  - *The Hunting of the Snark: An Agony in Eight Fits by Lewis Carroll*
  - *Jabberwocky by Lewis Carroll*
- `f_s_fitzgerald`:
  - *This Side of Paradise by F. Scott Fitzgerald*
  - *Tales of the Jazz Age by F. Scott Fitzgerald*
  - *The Great Gatsby by F. Scott Fitzgerald*
- `t_smollett`:
  - *The Adventures of Ferdinand Count Fathom — Complete by T. Smollett*
  - *The Expedition of Humphry Clinker by T. Smollett*
  - *The Adventures of Roderick Random by T. Smollett*

You can find the Gutenberg IDs by searching for the work and looking at the URL. The *number* after `/ebooks` is the Gutenberg ID. For example, Shakespeares *Macbeth* is `https://www.gutenberg.org/ebooks/1533`.

§ Task: Implement test-training set.

In the cell below the given cell with the contents of the test set (first line beginning with `## THIS IS YOUR TRAINING SET`)

- complete the list named `test_train_documents`.

The first item in the list will be the file containing all the Herodotus works loaded by the prior cells (e.g. `herodotus.txt`).

- fill in first four items of the list with the training set files **in this order**: Herodotus, Smollett, Fitzgerald, Carroll
- the last four items of the list, you will put the names of the training set files (e.g. those produced by the `test` list above.)

§ Task: Complete the `vectorizer`.

You will need to study the TF-IDF Vectorizer of ScikitLearn :

- sklearn.feature_extraction.text.TfidfVectorizer

You will then complete the implementation in the cell:

```
X = None
```

Hint: `X` should be "fitted" on the `test_train_documents` above. See also: `fit_transform()`.

§ Task: Interpret the output of the classifier.

After you have run the cell `clf.predict(X[4:])` you will see the list of numbers which represent the class labels from the training data.

Explain the output and answer the following questions:

- Would you say the classifier was accurate? Give a reason for your answer.
- What recommendation would you give to improve the classifier?

(30%) Learn about machine learning ethics.

With the increasing rise of machines and AI in human decision making and human activities, we are increasingly in need of critical conversations about the ethics of AI – arguably the conversation is incomplete around the ethics of data mining and data science writ large, so this conversation will act as a proxy and extension of that broader conversation.

You will listen the podcast *The Machine Ethics* podcast which covers wide ranging conversations about AI and Ethics and brings to the fore relevant conversations about machine driven decision making and the intersection with human beings. You will learn about "digital sociology" and the relevant impact this field has on how we might develop human-machine boundaries, especially in critical decision making spaces that intersect with human society.

Listen to **Episode #93** (October 3, 2024):

- "Socio-technical systems with Lisa Talia Moretti" / duration: 61m49s

You will need to absorb as much as you can and take notes. There will be an open note assessment on Blackboard, which will ask approximately 10 questions relevant to the talk, so it is best you actively listen to this fascinating conversation.

§ Task: Listen to the podcast and do the companion assessment.

Once you are done, there will be an online assessment about the podcast, which will be available on or after Dec. 5, 2024 through the last day of the final (which will be posted).