5243 Introduction to Data Mining

Assignment 1

Johns Gresham

# Problem Description

In an effort to build automatic article categorization, search, and graphs, the issue of comparing the contents of articles arises. To compare the similarity between two articles, we need a way to form measurements on the contents of the articles.

# Feature Vectors

### Features

For each article, a number of measurements will be contained in a feature vector that will aim to concisely capture the focus of the article. For example, words in the title, topics, or places section should be included in the feature vector. It seems likely that knowing two articles had similar titles, topics, or places would be helpful determining similarity. Also counting the frequency of each word in the article text could give insight to the main topics or subjects. Articles that mention ‘sales, ‘revenue, or ‘economy frequently might be categorized as a business article. Another common calculation to determine how unique a word is to a document is called tf-idf (term frequency-inverse document frequency). It is a value based on how frequent the word appears in the document and how common that word is in other documents. These two weights should give an overall feeling of how strongly a certain word is used in a document compared to the corpus of documents.

In the first program, *lab1\_1.py,* the word frequency is used. In the second, *lab1\_2.py*, the tf-idf value is calculated. Both of these features are applied only on words that seem useful. Stopwords are removed and words are lowercased and stemmed. Words that only occur once in the document are ignored. I describe the process I used to create the feature vectors on each article below.

### Processing Steps

First the meta data on the article, such as title, topics, places, are stored as a feature without much modification other than splitting words. Then the text of the article is cleansed. Using *word\_tokenize()* from the NLTK package the text is tokenized. Then stopwords and any token that only contains punctuation are removed using NLTK and regex respectively. Everything is lowercased and stemmed, so that capitalization or word endings don’t separate the frequency of the root word (ideally we consider ‘run’ and ‘running’ as two occurrences of the root word ‘run’). Now when referring to each word in the article, we mean each word remaining after the text is filtered and stemmed. Then the frequency of each word in the article is counted and the total number of words in the article.

In the first program, we are almost done. We just remove the words that don’t occur frequently enough to seem important. In our case, it is words that occur only once. Then the features are the metadata and the word frequencies. In the second program, we continue by calculating the tf-idf value for each word (filtered words that occur more than once), in each article. To calculate the tf-idf value of a word, the term frequency of the word in the document and the number of documents in the corpus containing the word is needed. The term frequency is just the number of times the term occurs in the document over then number of total number of words in the document, both of which we have already. All that’s needed is the number of documents in the corpus containing each word. This is done by iterating through each word in our filtered words for each document and incrementing a counter every time a document is found with the word.

# Results & Issues

The feature vectors tended to capture frequencies and tf-idf values for 5-10 words that were most often related to the main topic of the article. It took 5 minutes and 10 seconds for a single run on the first program through all the articles. This is about a tenth of a second for each article which seems reasonable. The second program which calculates the tf-idf value only took a few seconds longer than the first during my testing.

It was immediately obvious after looking at a few example output feature vectors that not all stopwords were removed that could be. For example, ‘said’ and ‘would’ were not removed and don’t indicate too much about the article. This could produce some noise and increases the feature vector size. And since only low frequency words are removed, these get included. Also, if the article text length was to increase, perhaps words that occur only 2 or 3 times would need to be removed. Right now only words that occur once are removed, which isn’t robust for differing article lengths. A minor note, a few extra features may have been calculated but not printed, such as lexical diversity and total number of words. These were experimental and might be used further on.

# File Locations

directory : /home/7/gresham/5243/lab1/

makefile: makefile

readme: readme

report: report.docx

first program: lab1\_1.py => outputs to: output\_1.txt

second program: lab1\_2.py => outputs to: output\_2.txt

first feature vectors for all articles: final\_output\_1.txt

second feature vectors for all articles: final\_output\_2.txt

The ‘final\_’ output files exist so all the feature vectors can be viewed without having to wait for them to be computed. Any running of the programs outputs to the ‘output\_X.txt’ files.