**Fall 2020 Research Project - NLP of Chinese Media**

Ryan Hebert, Dr. Jorge Silveyra, Dr. Chris Herrick

Goal: To analyze the text written by Chinese and American media about Hong Kong and Taiwan over time, looking specifically for trends of propaganda and sentiment.

# The Data

## Obtaining

Our project started out by attempting to pull data from the following sites:

* [Global Times](-https:/muhlenberg.idm.oclc.org/login?url=https://www.oriprobe.com/GlobalTimes_en.shtml)
* [People’s Daily](https://muhlenberg.idm.oclc.org/login?url=https://www.oriprobe.com/peoplesdaily.shtml)
* [China Daily](http://global.chinadaily.com.cn)

These sites store news in a dynamically paginated format, so a typical web scraper would not allow us to pull all the date from these databases. This pointed us to the direction of Selenium, a dynamic web driver that allows automated interaction. To use Selenium, you must first download the driver corresponding to your web browser. For our project, we used the ChromeDriver found [here](https://chromedriver.chromium.org/downloads). To implement this program, you must also download the framework using pip or conda. See full documentation [here](https://selenium-python.readthedocs.io/installation.html).

In the end we were unable to actually use this method due to 504 gateway errors when parsing the databases. We’d collect roughly 10k items, but before the program completed the server would stop responding and we’d lose our data. The new objective was to gain access to articles through Twitter. Using the framework [Tweepy](https://www.tweepy.org/) we were able to pull all the available Tweets from a specified user with their TwitterAPI wrapper. The return type of this call is a JSON file with objects for each tweet containing fields for date, likes, links, and the text body. From here, we pulled the links and stored them in a queue for requesting their HTML. It was important to look for links outside the domain of the target site to ignore for our goal. This code can be found in the tweetTimeline.py file.

## Cleaning

The next step was to separate individual files containing links to and body of the tweets and articles found from their tweets. To get the tweets, we simply used json deserialization and pulled the text tag contents. For the articles, we called a request to the link found in tweets using the Newspaper3k framework to get just titles, dates and body text. The documentation for this framework can be found [here](https://newspaper.readthedocs.io/en/latest/). The code for this process can be found in the urlExtractor.py and articleDownloadScript.py files.

## Training

Early on we found a research project called Proppy that had a similar goal of categorizing text in terms of Propaganda and non-Propaganda. The full paper can be read [here](https://arxiv.org/abs/1912.06810). The data that the developers on this project used is open source (see their GitHub). To test our models, we split this data and saved it in the file train50.csv with a roughly 50-50 data set in terms of annotations.

# The Models

## Vector Space Model

Implementing a VSM was not as successful as other methods, but it was still a good learning experience. Instead of using NLTK or other libraries, we implemented a cosine similarity algorithm from scratch. The code and implementation are in the vsm.py file. This also has methods for our training calculations as well, which we found did not correctly predict any propaganda.

The VSM operates by placing each file of our training data on a graph as a vector. Consider the axis of the variables arbitrary but consistent, so vectors closer together represent documents similar to each other. Using the data obtained from Proppy, we created our inverted index of words in two separate files. The first being a dictionary of string (key) and list of int (value). The key represented each word found in each of the documents, and the list value contained each number file that the word appeared in. This allowed us to find our document frequency. For an example, see proppyDocuments.json in the Model directory. The second JSON file had the format of int (key) dictionary (value), where the sub dictionary had the format of string (key) int (value). The sub dictionary represented each word and how many times it appeared. This was only for the document currently being evaluated, shown by the outer dictionary key. See proppyOccurancy.json in the Model directory.

For more information on calculating the TF-IDF score, visit [here](https://iyzico.engineering/how-to-calculate-tf-idf-term-frequency-inverse-document-frequency-from-the-beatles-biography-in-c4c3cd968296), the tutorial I used to perform these calculations. The code for this solution is in amazonBot.py. Again, this part of our project was unsuccessful and has thus not been updated since discovering its lack of use. This model is designed less for document classification as it is for similarity, so we used the KNN approach to determine the test document’s K nearest neighbors and assumed it had the same classification as the majority of that K. All results yielded a 0, denoting non-propaganda, so we did not feel comfortable giving this model an appropriate accuracy score.

## Support Vector Machine

SVM algorithms differ from VSM algorithms. Support Vectors are like boundaries, calculated similarly to the vectors from the previous model. We calculate these in some way to numerically represent each document and given all of the calculations and their annotated classifications, the model identifies support vectors which will act as boundaries. Anything between these vectors will share a classification, while those outsides would denote another classification.

*For further clarification on the algorithm and some images, the following information comes from Dr. Silveyra’s 2021 Artificial Intelligence course lecture:*

This is a very complex statistical method to divide data into different categories. Fortunately, we will not have to manually calculate (phew) the values, but we still need to understand what it is. SVMs are similar to the previous learning methods that we learned, but they can be

expanded to include linear, non-linear, and unsupervised classification. To understand the basic model, let's look at the following image:

Chart, diagram

Description automatically generated

On the left image, we can notice that the main concept is similar to the previous methods we learned (finding a "line" that separates items of different categories). However, the main issue is that all of those lines are "arbitrary" and potentially not optimal. Meaning that those lines are designed for these particular values (overfitted) and will not be easy to introduce new values. Instead, with SVM we have an "optimal" line (plane) that divides the two categories. This is possible by finding the "last" points from each category that can help us separate the different categories (called vectors). Not only finding the vectors are enough, but we also want to find the line (plane) that has the optimal middle distance between the vectors.

## NB

The Naïve Bayes algorithm can be used to classify text using Bayes’ Theorem. “In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as ‘Naive’.” ([Source](https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/)) See the following formula to better understand the probability calculation.

Text, letter

Description automatically generated

## Sentiment Analysis

Sentiment is roughly defined as an attitude to a situation or event. Analyzing the sentiment of a word is, for the most part, an easy and useful tactic to make generalizations about the writer or speaker. When analyzing sentiment of larger data sets this process gets trickier. Consider the data set “I like people”. If we were to score each word based on its’ sentiment and average the scores to give the data set a total score, the process would look something like [0, 1, 0] in which 0 refers to neutral, 1 to positive, and -1 to negative and where each index refers to the respective token. Thus, our data set would have a sentiment of 1/3 or .334, a positive number representing a positive data set. If we were to slightly modify the sentence to “I like hurting people”, our calculation would change to [0,1,-1,0], producing a score of 0 denoting a neutral statement. Anyone who speaks English knows this is a negative sentence, so in order to correct this we use n-grams. Bi-grams, or groupings of 2 words together, would change the sentence to being analyzed as “I”, “I like”, “like hurting” “hurting people” and “people”. These pairings of words have different scores than the sum of their parts, generating more accurate analysis on the sentiment.

In our project, we used the Vader library for analyzing sentiment. This used a combination of n-grams to get the most accurate results possible. The following snippet from textclassify.py shows this calculation.

Text

Description automatically generated

# Visualization

## Bokeh

Bokeh is an open-source data visualization library documented [here](https://docs.bokeh.org/en/latest/index.html). We chose this because it offered interactive plotting servers that auto-generate on completion of our core process. Their Gallery gives several code examples of in-depth visualization programs that helped build outs. Although basic at this point, there’s a lot of potential to build these models into high quality, interactive figures that can live on a website. Visual.py and visual2.py are yearly and monthly chart programs that can be called from the command line using the following call, replacing myapp.py with the name of your file:



[Installation](https://docs.bokeh.org/en/latest/docs/installation.html)

## GUI

Our User Interface was the least complete part of this project. Due to time constraints, this was the lowest priority and is likely the next step for this project. What we have so far is a window that. Calls on the Mac file explorer to choose a data set. This must be a comma or tab separated file with the uncleaned data and date in columns A and B. What should happen is a Process is started that runs through our full sequence (clean -> predict -> summarize -> visualize). Another idea we had that was never implemented was a configuration UI. Being able to specify whether or not or how to clean the data, what variables to show in the graphs and other toggles would be useful.

A non-working program of our UI is found on gui-test.py and can be a good place to start. Using tkinter (documented [here](https://docs.python.org/3/library/tkinter.html)) my goal was to have a simple UX where the choices are to analyze the propaganda and sentiment of either a document or a folder of documents. The config UI could either be on the main page or separate but is currently in buttons under the dialog box for folder selection. Those buttons do not currently work. The next step is to link the “Run” button to creating a Process object and running it.

If at any point help is needed explaining my code, please reach out to persona email @ [rrheb31@gmail.com](mailto:rrheb31@gmail.com) or ask Jorge to reach out to me.