Brian Farrell

Big Data Exercise 4

Page Rank Question

Q1-4. [16 points, *answer should be in your PDF submission*] An alternate measure of economic rank of an area might be the inverse of a "hardship index," defined and quantified here. How well, or poorly, do your calculations of TrafficRank for Chicago community areas correspond with the inverse of hardship index? Give *a qualitative analysis in plain English*.

- The results of my page rank correspond well with the claim that page rank has an inverse relationship with hardship index. The two areas that have the highest page rank, 8th and 32nd, have some of the lowest values on the hardship index. The hardship index values are 8.6 for 8th and 9 for 32nd. These are very low values for this index and are also the two lowest values on the index as well. The inverse appears to be less true. The two lowest traffic rank values are 1st and 48th. These two areas have hardship indexes of 39.4 and 38.4 respectively. While these hardship indexes are much higher than the other two examples, they are roughly around the middle of the index. There are much higher hardship index values than these two areas, despite the very low traffic rank. This shows that my traffic rank data is much better at predicting low hardship indexes than it is at predicting high hardship indexes.
- There are some other factors that could cause other problems with this new measure. Locations of airports and tourist destinations in a city could disproportionately affect the traffic rank of a certain area, regardless of the hardship index score. Tourists would travel to where the sights are regardless of the hardship index value says, so they could

potentially skew the results. In a similar fashion, people go to the airport regardless of the level of hardship, so this could be another potential factor skewing results. If the airport is in an area with average or higher hardship index value, it would likely have a very high traffic rank just due to taxis going to the airport.

Presidential Speeches Questions

Q2-1. [4 points, *answer should be in your PDF submission*] We can calculate TF values of each word in a given document. Explain why the calculation of IDF can only apply to a corpus, not to a specific document.

- You need to be able to categorize the specific collection of documents first before analyzing the words in them. With a single document, you would not be able to determine which words appear often in a certain category of document unless you had other documents in the same category to compare with. For example, a baseball article about the Boston Red Sox might mention the "Green Monster" a few times if it happened to play an important role in the game the article was covering. Analyzing this singular article would result in the "Green Monster" being selected as an important word (or phrase in this example) for baseball when it is only important to the Boston Red Sox. However, a collection of baseball articles would help avoid this problem because out of a large collection, the "Green Monster" would be mentioned very little compared to the single article.

Q2-2. [6 points, *answer should be in your PDF submission*] The implementation of Scikit-Learn's IDF calculation differs from that of the "standard" calculation. What is the justification for this change?

- The +1 change to the numerator and denominator in how Scikit-Learn calculates IDF is added to avoid a potential division by zero. If a given word does not appear in any document in the list, the denominator would be 0, leading to division by 0 in the original equation. Scikit-Learn adds 1 to the denominator to avoid this edge case. One is also added to the numerator to keep the proportion consistent.
- The reason for the change with the +1 at the very end of the formula was for the case where a word is mentioned in every document. According to the general formula, this case would result in an IDF value 0. The formula is adjusted to give the word a value in this case, since it does appear in the documents.

Q2-3. [25 points*, answer should be in your PDF submission*] Each president has a .tar.gz file containing his speeches. Filter the text using remove_stopwords(bow) code provided, where bow is the bag of words in the text. Write a Python procedure to calculate TF.IDF for any president's speeches and output the top-15 most important words in their speech.

- The top-15 words for Abraham Lincoln's speeches and their respective scores are pasted below:
    - states: 0.1522
    - government: 0.0975
    - people: 0.0932
    - union: 0.0897
    - slavery: 0.0785
    - united: 0.0739
    - war: 0.0700
    - state: 0.0687

    - constitution: 0.0660
    - congress: 0.0617
    - great: 0.0580
    - men: 0.0551
    - country: 0.0536
    - year: 0.0445
    - free: 0.0441
- The complete code (if necessary) is pasted below:

```
import os

import re

import requests

from sklearn.feature_extraction.text import TfidfVectorizer

# Given code

stop_url =

"https://gist.githubusercontent.com/rg089/35e00abf8941d72d419224cfd5b5925d/raw/12d899b70

156fd0041fa9778d657330b024b959c/stopwords.txt"

stopwords_list = requests.get(stop_url).content

stopwords = set(stopwords_list.decode().splitlines())



def remove_stopwords(text):

    words = re.sub(r"[^a-zA-Z0-9]", " ", text.lower()).split()
```

```
    return " ".join([word for word in words if word not in stopwords])


def load_speeches(directory): #method to get all the speeches for a specific president
    texts = []
    for root, _, files in os.walk(directory):
        for file in files:
            if file.endswith(".txt"):
                with open(os.path.join(root, file), 'r', encoding='utf-8', errors='ignore') as f:
                    texts.append(remove_stopwords(f.read()))
    return texts


def compute_tfidf(speeches):#method to compute the tfidf for the speeches
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(speeches)
    feature_names = vectorizer.get_feature_names_out()


    avg_tfidf = tfidf_matrix.mean(axis=0).A1
    word_tfidf = dict(zip(feature_names, avg_tfidf))
```

```
    return sorted(word_tfidf.items(), key=lambda x: x[1], reverse=True)[:15]


if __name__ == "__main__":
    lincoln_folder = "lincoln"  # Chose lincoln


    speeches = load_speeches(lincoln_folder)
    top_words = compute_tfidf(speeches)


    print("Top 15 most important words in Lincoln's speeches:")
    for word, score in top_words:
        print(f"{word}: {score:.4f}")
```

Q2-4. [15 points*, answer should be in your PDF submission*] Examine the result carefully – at least ***some*** of the top TF.IDF words should be historically consistent with what was going on in the country at the time. You just have *very slight* control over the outcome (by adding more words to the initial set of stopwords).

- The collection of words is historically consistent with the time period Abraham Lincoln gave these speeches. A lot of words, like free, country, year, great, men, congress, constitution, states, government, people, union, united, could be words that would be present in any president's speeches. However, slavery is a word that was historically extremely important to Lincoln's presidency. Some of the typical words could also be

used in a unique context in Lincoln's speeches compared to their use in speeches given by other presidents. Union is a word that could be used to describe the country in any period, but it was especially relevant to Lincoln because of the federal government calling their troops the Union Army. Another example are the words free and state in the top 15 words. Freedom is a big theme of presidential speeches throughout US history, but it is uniquely dominant in speeches during the Civil War. These terms likely could be describing the enslaved people being freed, or the distinction between free and slave states. Overall, the top 15 words appear to line up with the historical context of Lincoln's presidency.