APPLICATION OF TOPIC MODELING ON POLITICAL COMMENTARY

A Project Presented to

The Department of Electronics and Computer Science

At the University of Missouri



In Partial Fulfilment

of the Requirement for the Degree

Master of Science



by

ADITYA PARASHAR

Dr. Jeffrey Uhlmann, Project Supervisor

Dec 2017

Contents

[LIST OF FIGURES & TABLES 4](#_Toc499490925)

[ABSTRACT 5](#_Toc499490926)

[Chapter 1 6](#_Toc499490927)

[1. Introduction 6](#_Toc499490928)

[1.1. Motivation 6](#_Toc499490929)

[Chapter 2 9](#_Toc499490930)

[1. Atmosphere - Cyverse Resources 9](#_Toc499490931)

[2. Related Work 10](#_Toc499490932)

[Chapter 3 14](#_Toc499490933)

[1. Question to solve 14](#_Toc499490934)

[2 Environment Setup 15](#_Toc499490935)

[2.2 Collecting Data 16](#_Toc499490936)

[2.3 Document Statistics 17](#_Toc499490937)

[2.4 Tf-idf Algorithm 17](#_Toc499490938)

[Chapter 4 21](#_Toc499490939)

[Experiments conducted 21](#_Toc499490940)

[1. Training the Tf-Idf model 21](#_Toc499490941)

[2. Evaluating similarity 22](#_Toc499490942)

[3. Combining and collecting results 23](#_Toc499490943)

[Results 23](#_Toc499490944)

[1. FOX CORPORA 23](#_Toc499490945)

[2. MSNBC Corpora 24](#_Toc499490946)

[Chapter 5 27](#_Toc499490947)

[Summary 27](#_Toc499490948)

[Further work 28](#_Toc499490949)

[Bibliography 29](#_Toc499490950)

[Appendix 30](#_Toc499490951)

# LIST OF FIGURES & TABLES

Figure 1: Target topic – similar topic

Table 1**:** Comparison of Cyverse with non-hosted environments

Figure 2: Classifier and Predictor: Word vector concatenation and prediction

Table 2: Gensim dependent packages

Figure 3: Workflow for Topic Explorer

Table 3: Document Statistics

Figure 4: Runtime comparison for Generative algorithms vs Statistical algorithms

Figure 5: Tf-Idf model trained based on vector size.

Table 4: “sims” index1

Figure 6: Results of similarity with Fox Corpora

Table 5: “sims” index2

Figure 7: Results of similarity with MSNBC Corpora

# ABSTRACT

Natural Language Processing (NLP) has been used across different disciplines to process large amounts of data but using NLP alone is not sufficient to understand, analyze, distinguish and categorize textual data. Specifically, in instances of political discourse, there are a lot of similarities and subtle differences in text. This project describes and implements one such algorithm - Term Frequency-Inverse Document Frequency (Tf-Idf) and is then applied on two distinctly identified political commentary narratives.

Implementing Tf-Idf, this project views well defined text topics and finds similarity scores from the commentary narratives. In addition to Tf-Idf this project utilizes deep learning via Word to Vector models and categorizes data sets.

Deep learning as such has been used across multiple different areas to learn and work on tagged datasets. Here we used deep learning models to determine the similarity of testing corpora on a trained model.

The focus of this project is to create a model trained on political commentary and to identify whether this model can pinpoint the similarity of non-trained data with political discourses to aid understanding of such discourses. Additionally, an attempt to determine if specific implementations of deep learning could be used to further help the current understanding of political thought processes. The deep learning API that was used was – Gensim. It was used to understand, evaluate, create and compare topics from political commentaries.

# Chapter 1

## Introduction

## Motivation

Machine Learning (ML) is used across multiple areas and fields to find patterns and train models. The trained models are eventually used to understand and derive high quality information and make predictions on data. Additionally, ML algorithms can be used across images or text. Images are inherently ambiguous, words however are semi-structured and they also contain information about themselves. Which makes working on text much easier since the metadata helps identify, structure and compare the text. That said, textual analysis has presented many challenges for machine learning too. One of the biggest disadvantages is the laborious and manual feature extraction from text.

The most widely used methods in text analyses include Word to Vector (Word2Vec), Latent Dirichlet Allocation (LDA) and Latent Semantic Indexing (LSI). All of these methods use a procedure where each document is considered to have a set of topics that are assigned to it via LDA and the algorithm determines the relative distance or proximity of those documents. The vector distances are then compared to measure similarity or diversity of the documents.

On my current project I used political commentaries to find out how similar they were. This was done using a baseline of two distinctly disparate corpora. Finding out similarity scores against ground truth helped to identify what the parameters were critical in identifying the similarity.

For this project, “Genism” was used to work on text data. Gensim is a tool that is used to realize unsupervised semantic modeling from plain text. Gensim, as an API, provides the ability to use term frequency using Information Retrieval (IR) to measure vector distances.

A simple example which details the process of going about finding similarity is in Figure 1. When a target topic from the training data is identified, similarly weighted topics from the test data is used to find the importance of the topics and ascertain the similarity scores.

*Similar topic*

***WORK***

Technology

*Similar topic*

***MAGIC TRACKPAD***

Technology

*Similar topic*

***XCODE***

Technology

*Target topic*

***BASKETBALL***

Gym Class

*Similar topic*

***SWIFT C#***

Technology

*Similar topic*

***IPAD***

Technology

Figure 1: Target topic – similar topic: If the target topic that is identified is MACBOOK, similar topics that result from the model and the overlap and the importance of topics are IPAD, SWIFT, MAGIC TRACKPAD and XCODE.

# Chapter 2

## Atmosphere - Cyverse Resources

Atmosphere is a cloud-hosted environment specifically designed for use by research and doctoral students. For this project, a cloud-hosted Unix environment which had 8 GB of RAM and a single core processor was used. The python library was ported to an x64-based processor and the hosted environment provided pre-defined images which could be started and suspended as needed.

Also, the installed images of Ubuntu were present on multiple servers. This configuration provided a heterogeneous environment which had multiple general and specialized machines.

Table 1**: Comparison of Cyverse with non-hosted environments**

|  |  |  |
| --- | --- | --- |
| **Config / Parameter** | **Non-hosted Environments** | **Cyverse** |
| Memory Management | Fixed and built into hardware. x86-64, x86 | Managed by Atmosphere, uses a memory controller. |
| Virtual Memory | CPU specific. | Segmented by physical memory. %age wise is greater than fixed memory. |
| Processors | I5, i7 Octa Core processors. | Intel   i7 900 series |
| Memory Page Interrupts | Might need reboots to map pages to processes | Managed by the host – atmosphere. |

## Related Work

The main goal of this project was to determine how similar political commentaries were. Semantic similarity is a widely studied and analyzed research problem. The previously studied methods predominantly start out with a generic model and become tailored to suit specific contexts.

The semantic usage of words and how words present in a document determine the importance of each word. Word Maturity Model [1] expresses the maturity of a word as the function of two

Parameters: word “w” and age “e”. Given a word w and an age e, the function estimates the maturity of the word at that age, and also establishes when that word has a mature representation. This model computationally models meaning, maturity and quantitatively evaluates the quality of the text on the basis of the model. This methodology, called Word Maturity (WM), models the maturity of the lexicon by analyzing the trajectory of a word's representation at several points in time. The WM model was based on the premise that word meaning varies with age in a continuous manner. This model makes no sense to talk about the meaning of a term as something static and absolute, but rather it would develop as it is exposed to new lexical contexts.

To add to lexical contexts there has been a significant amount of work done which involves taking the order and the structure of the words present in the corpora. To add to classifiers there is also research on learning based on structure. A well-known( ??) model which involved training the classifier to pick up on indexed positions of words predicts the next word in a sentence. In this framework, every word is mapped to a unique vector, represented by a column in a matrix W. The column is indexed by position of the word in the vocabulary. The

Concatenation or sum of the vectors is then used as features for prediction of the next word in a sentence.

Classifier

on

Average / Concatenate

Word Matrix

W

W

W

W

sat

cat

the

Figure 2: Classifier and Predictor: Word vector concatenation and prediction based on uniqueness and positions of vectors

Some of the biggest drawbacks is that techniques like word vectors only work on sentences, but not paragraphs/documents with several sentences. It is unclear how to combine the representations over many sentences. Such techniques therefore are restricted to work on sentences but not paragraphs or documents.

On the flip side there is also significant research and results that come out of models like Diffusion Theory [3] which looks describes the spread of product opinions in social systems. According to this theory, not only the product characteristics (e.g. complexity) are an important factor in the formation of opinions but also the structure of the social networks. This theory communities comprises four steps. In the first step the users’ opinions on the product are extracted by text mining. In the second step, the communication relationships among users are identified by text based relationship mining methods. The extracted users, opinions, and relationships form a social network which is represented as graph. Nodes and edges of the graph can be characterized by attributes. The nodes represent the users of a forum and the edges their communication relationships. Models like these help explain the flow of information and thought across social networks.

More recently there has also been hybrid approaches which involves a mix of paragraph vectors[a] and LDA. Our new method as shown in projects words, documents, and topics in a high-dimension semantic space. A document vector is considered as a single vector, which is the centroid of all words in the document as what Word2Vec does in the projection layer. In addition, each document has its individual length, thus its vector is divided by the number of words in the document to guarantee the measurements with same scale.

The Hybrid Document Feature Extraction Method[b] creates topics where a subset of high-probability words in each topic is employed to represent the topic, and then their probabilities are rescaled as the weights of words. Hence different words have different contributions to the topic.

Apart from the contributions the topics also have a semantic relatedness to each other. There is distinct difference that is explained in [4] where “Cars and gasoline would seem to be more closely related than, say, cars and bicycles, but the latter pair are certainly more similar.” The key derivative from Budanitsky was the concept of Semantic Distance “A” simple way to compute semantic distance in a taxonomy is to view it as a graph and identify relatedness with path length between the concepts. The semantic distance is the path from one node to another. The number of edges between terms is a measure of conceptual distance between terms”. Despite the simplicity of this distance function, the ability to obtain surprisingly good results in information retrieval task is what makes it so widely used. As is the case with vector similarity following assumptions were made by Budanitsky:

1. The similarity between arbitrary objects “A” and “B” is related to their commonality; the more commonality they share, the more similar they are.

2. The similarity between “A” and “B” is related to the differences between them; the more differences they have, the less similar they are.

3. The maximum similarity between “A” and “B” is reached when “A” and “B” are identical, no matter how much commonality they share.

# Chapter 3

### Question to solve

The use of Natural Language Processing (NLP) to understand human language has been used across multiple areas. Specifically, this includes an understanding of how transcripts and commentary involve some context across a larger number of corpora. This is especially true in the case of political commentary where NLP can be used to understand often opposite and disparate views on the same topic. There are very few applications that specifically look at what differentiates opposing views in political reports. The project is further motivated by the need for determining similarity between small pieces of text across documents that potentially span different topics during multi-document summarization.

For this project, we define Similarity to be an index where the query vector is equivalent to the document corpora.

* 1. An OH-58 helicopter, carrying a crew of two, was on a routine training orientation when contact was lost at about 11:30 a.m. Saturday (9:30 p.m. EST Friday).
  2. (b) “There were two people on board,” said Bacon. “We lost radar contact with the helicopter about 9:15 EST (0215 GMT).”
  3. (c) An OH-58 U.S. military scout helicopter made an emergency landing in North Korea at about 9.15 p.m. EST Friday (0215 GMT Saturday), the Defense Department said.

We consider units (a) and (b) in Figure 1 to be similar, because they both focus on the same event (loss of contact) with the same primary participant (the helicopter). On the other hand, unit (c) in Figure 1 is not similar to either (a) or (b). Although all three refer to a helicopter, the primary focus in (c) is on the emergency landing rather than the loss of contact.

## 2 Environment Setup

Atmosphere provided an Ubuntu server computing resource. Since Gensim, as a Natural Language processing package, requires python 2.7, all the packages installed were for python 2.7. The dependent packages that were also installed include the following:

Table 2: GENSIM DEPENDENT PACKAGES

|  |  |
| --- | --- |
| Package Name | Version |
| gfortran | 7.2.0 |
| libopenblas-dev | 0.2.19-3 |
| liblapack-dev | 3.2 |
| scipy | 1.0.0 |
| numpy | 1.13.3 |

Gensim was used as the package since it is the most widely used topic modeling API. As a library, it has multiple deep learning models which can be used and implemented. Gensim and tensor flow as libraries are the most advanced and extensively used libraries. The wide adoption of the modeling API also provides support and maintenance of the implementation.

## Collecting Data

The data for this project was taken from transcripts that were freely available on news websites. The data was extracted as text files into labeled corpora. To extract just the raw text, the text files were processed using a data cleansing operation. Since the algorithm takes in documents as strings there were a few pre-processing steps that had to be done to ensure that the documents were indeed ready to be analyzed.

TOPIC EXPLORER WORKFLOW

**launch**

co

cor

**train**

co

cor

**prep**

co

cor

**init**

co

cor

**notebook**

**co**

**cor**

Corpus-model.npz

co

cor

Corpus-rev.npz

co

cor

Notebooks

co

cor

Corpus.npz

co

cor

Figure 3: Workflow for topic explorer

1. Saving the commentary as “.txt” files to ensure they could be read into UTF-8 encoded files
2. Removing carriage returns and newlines
3. Removing special characters – asterisk(\*), single quotes (‘), double quotes (“) and ticks (`)

From the two websites, the above four characters were the only ones which demarcated the text as invalid corpora.

## Document Statistics

All the commentary which was analyzed was obtained from the month of July 2017 through until October 2017. The following tables provides the details of the documents present in the corpora.

Table 3: Document Statistics

|  |  |
| --- | --- |
| **Feature** | **Statistic** |
| Number of documents | 48 |
| Avg. Number of words per document | 1600 words |
| Total Number of words | 76800 words |

## Tf-idf Algorithm

The most commonly used algorithm to analyze textual corpora is the Term Frequency – Inverse Document Frequency algorithm. Tf-idf is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The tf-idf value increases proportionally to the number of times a word appears in the document and is often offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general. It is a simple transformation which takes documents represented as bag-of-words counts and applies a weighting which discounts common terms (or, equivalently, promotes rare terms). It also scales the resulting vector to unit length Tf-idf is one of the most popular term-weighting schemes. The TFIDF function weights each vector component (each of them relating to a word of the vocabulary) of each document on the following basis. First, it incorporates the word frequency in the document. Thus, the more a word appears in a document (e.g., its TF, term frequency is high) the more it is estimated to be significant in this document. In addition, IDF measures how infrequent a word is in the collection. This value is estimated using the whole training text collection at hand. Accordingly, if a word is very frequent in the text collection, it is not considered to be particularly representative of this document (since it occurs in most documents; for instance, stop words). In contrast, if the word is infrequent in the text collection, it is believed to be very relevant for the document. TFIDF is commonly used in IR to compare a query vector with a document vector using a similarity or distance function such as the cosine similarity function.

Tf-idf involves the product of two statistics:

* + 1. Term frequency: In the case of the term frequency tf(t,d), the simplest choice is to use the raw count of a term in a document, i.e. the number of times that term t occurs in document d.



Where:

tf(*t*,*d*) = term frequency

*ft*,*d* = raw count of the number of terms in the document

t = term in the document

d = document

2.4.2 Inverse document frequency idf is a measure of how much information the word provides, that is, whether the term is common or rare in all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word, obtained by dividing the total number of documents by the number of documents containing the term and then taking the logarithm of that quotient.



Where:

N = total number of documents in the Corpus, N = |D|

The above algorithm was implemented using the gensim API. Following were the steps followed to identify similarity in the corpora:

1. The ground truth for similarity was one corpus of Rachel Maddow and Sean Hannity each.
2. The baselines were filtered and the pre-processing steps were performed on each of the ground truths.

The model was trained on the ground truth.

1. After which the corpora from MSNBC and Fox were preprocessed.
2. The final step involved using the above pre-processed test corpora to evaluate the similarity index.

Rationale behind Using Tf-Idf:

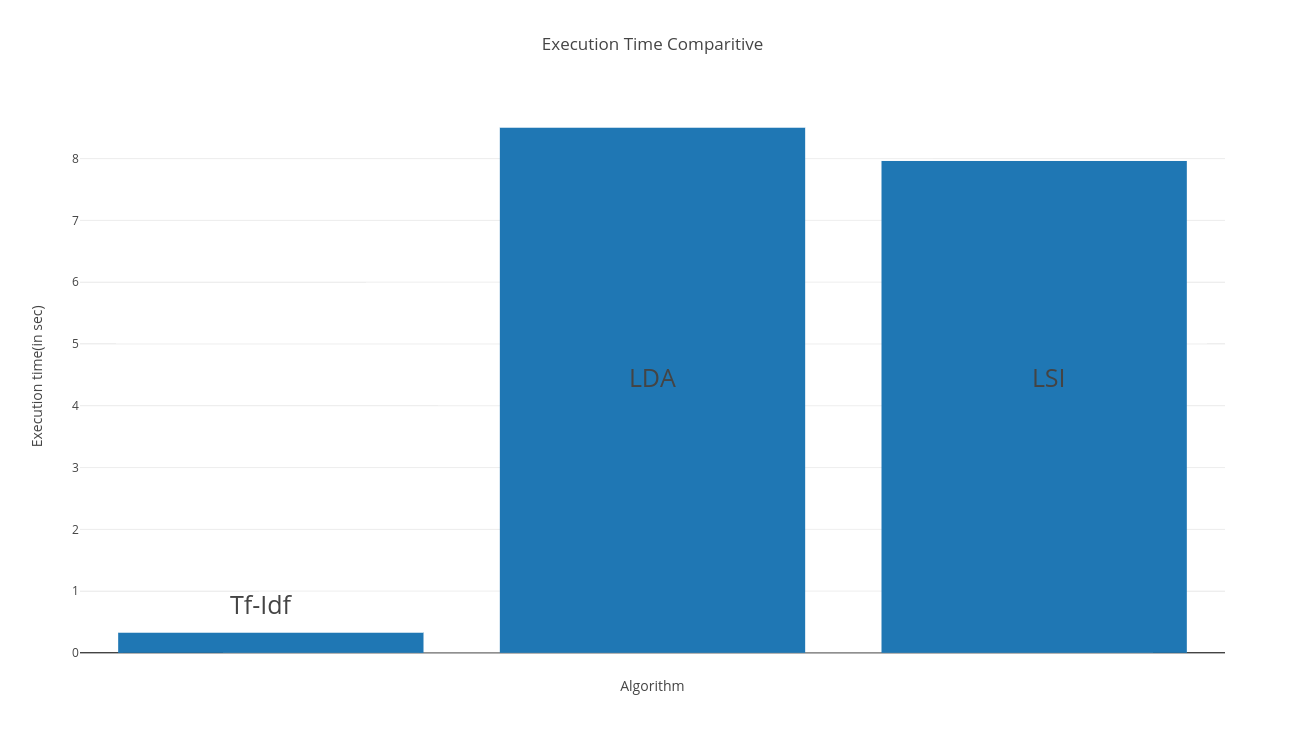


Figure 4: Comparative runtime comparison for Generative algorithms vs Statistical algorithms

Tf-Idf as an algorithm takes significantly lesser time than either of the generative models( LDA and LSI). Since the purpose of the project was classification and since the approach was heurisitic this model of information retrieval was effectiive. More importantly this weighting method leverages the information implicitly contained in the categorization task to represent the document. Results : how fox contributors are pretty much the same. But the msnbc ones are not as similar. And also how contributors dictate the similarity more than the host.

# Chapter 4

## Experiments conducted

After the data, cleansing and pre-processing operations the experiments that were conducted were split up into three distinct parts:

## Training the Tf-Idf model

The raw documents which had the stopwords removed were parsed into data structures. The data structures were arrays and were encoded with the UTF-8 text format, so we decided to leave in the formatting tags to account for noisy data and to test the robustness of TF-IDF. We simulated more noise by enforcing case-sensitivity.



Figure 5: Tf-Idf model trained based on vector size. From the graph, the greater the vector size, the similarity score and the extraction is greater.

## Evaluating similarity

After the model has the initial training the files themselves are loaded in memory into the corpus attribute. In order to perform any we first need to turn the text content into numerical feature vectors. The corpus attribute converts the FOX and MSNBC text files into a dictionary object. The dictionary is then converted into feature vectors.

Gensim’s API has the option of passing in multiple parameters into similarity comparison. The Similarity class splits the index into several smaller sub-indexes (“shards”), which are disk-based.

Atmosphere’s ability to utilize all of the available memory and keep the shards in memory made the response to be much faster. In comparison the same similarity comparison on a windows machine took 8x speed.

The feature vectors are in turn saved in an attribute called “query\_doc”. Subsequently the query\_doc is fed into the Tf-Idf model. And the similarity index is invoked on the Tf Idf model.

The similarity index uses a factor called “chunksize”. Which is basically a sequence of floats to identify the number of splits in the corpora. The size of chunksize is a tradeoff between increased speed (bigger chunksize) vs. lower memory footprint (smaller chunksize). If the distributed mode is on, each chunk is sent to a different worker/computer.

Finally the similarity index – “sims” is evaluated based on query\_doc’s tf-idf model.

## Combining and collecting results

The results of “sims” was a pair wise comparison. Each of Hannity (FOX) and Maddow (MSNBC) corpora were compared with the test data. The test data that was used were Lawrence O Donnell on MSNBC and Brett Briar on FOX.

The “sims” attribute gave out results comparing Hannity with O’Donnell (MSNBC) and Briar (FOX) and Maddow with O’Donnell (MSNBC) and Briar (FOX).

## Results

## FOX CORPORA

The “sims” attribute returns results as a comparative score. The first conclusion was that the greater the number of corpora the comparative score went up significantly. The results for both corpora went up. However the similarity of corpora for FOX went up much more significantly as the corpus size increased. Indicating that the greater the number of vectors present in the tf-idf model the similarity score increased at a much faster pace.

|  |  |
| --- | --- |
| **Corpus Size** | **FOX** |
| 5 | (42.875%, 13.53% ) |
| 10 | (47.167%, 14.992% ) |
| 15 | (55.003%, 15.274% ) |
| 20 | (73.0824%, 16.871% ) |
| 25 | (79.359%, 17.897% ) |

Table 4: “sims” index showing similarity scores when Hannity, Maddow were compared with FOX News Corpora

The scores were determined purely based on increasing size. Since the process involved trying to understand the tf-idf model, this was a heuristic analysis.

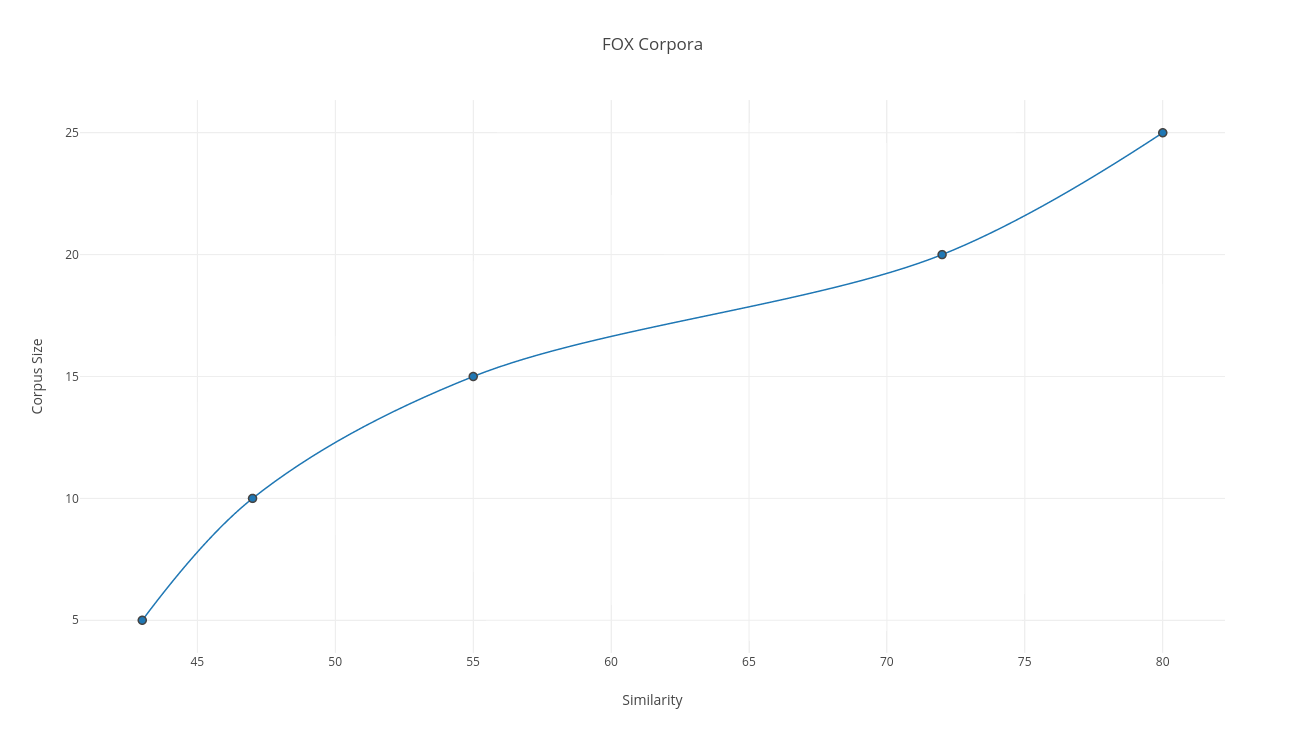


Figure 6: Results of the similarity with Fox Corpora (Sets of 5 corpora were used)

## MSNBC Corpora

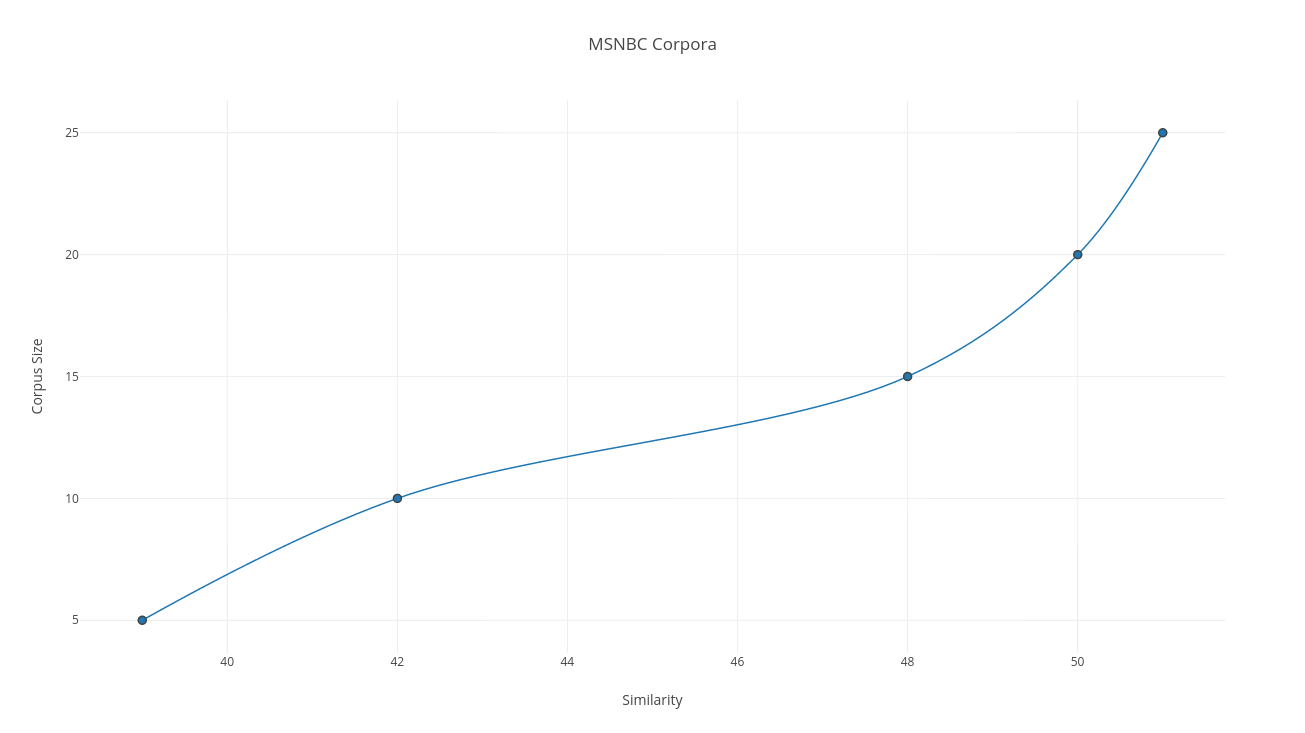
The results from MSNBC corpora when compared with Maddow and Hannity did not seem to show similar amounts of similarity. The percentage increase when the corpus size increased too, did not seem to show increased likeness.

Table 5: “sims” index showing similarity scores when Hannity, Maddow were compared with MSNBC News Corpora

|  |  |
| --- | --- |
| **Corpus Size** | **MSNBC** |
| 5 | (37.482%, 7.881% ) |
| 10 | (42.025%, 8.367% ) |
| 15 | (47.993%, 10.008% ) |
| 20 | (50.0134%, 13.176% ) |
| 25 | (52.739%, 17.556% ) |

High values of are concentrated on the end of the graph, so basing query retrieval on the top words here will likely return relevant documents.

Figure 7: Results of the similarity with MSNBC Corpora (Sets of 5 corpora were used)



This could indicate that the resultant vectors that come out of Hannity and Maddow do not seem to match with the MSNBC corpora. However the MSNBC corpora comparatively also seem to have greater correspondence with Hannity than FOX Corpora with Maddow. This is an aspect which was looked at too. Though the purpose of the project was not to find out if the cross similarity is of any significance at all.

For the cross similarity, the number of topics resulting from Hannity and Maddow was compared against the MSNBC corpora. The topic overlap for Hannity and MSNBC seemed to be slightly greater than the topic similarity for Maddow and FOX.

The graph above gives the confidence with which the corpora are categorized as.

# Chapter 5

## Summary

Automated measurement of the similarity between text documents is fundamentally a psychological modeling problem. This paper presents an assessment of keyword, n-gram and LSA approaches against human data for a small corpus of short news documents. In this project there were assumptions made which included that stopwords be removed and that the effect that it would have on the similarity would be insignificant. It is clear that when the model judges two documents from the FOX Corpus to be highly similar, it is correct. However there is only a 50% likelihood of the similarity with the CNBC corpora.

Aside from the similarity that arose out of using the Tf-Idf algorithm, it shows that as the model gets more data and more text the similarity goes up. When examining the words in the query, we see that TF-IDF can find documents that make frequent use of said words and determine if they are relevant in the document. The discriminatory power of TF-IDF allows the retrieval engine to quickly find relevant documents that are likely to satisfy the user. The model itself was trained on corpora which were from the month of August for both FOX and MSNBC, which makes the model to be data agnostic.

Both the FOX and MSNBC results had the same number of training examples. Also there was no background information that was fed to the model itself.

The most obvious is that the lower the number of corpora, the similarity that results from it is also low. While the Fox news similarity increased as the size of corpus increased, the same was not the case with MSNBC. This gives empirical evidence that the greater informative the training data is, the greater the advantage in having a corpus of background knowledge available for use during classification.

## Further work

Since TF-IDF is merely a staple benchmark, numerous algorithms have surfaced that take the program to the next level. (Berger et al, 2000) propose a number of these in a single paper, including a version of TF-IDF that they call Adaptive TF-IDF. This algorithm incorporates hill climbing and gradient descent to enhance performance. They also propose an algorithm for performing TF-IDF in a cross-language retrieval setting by applying statistical translation to the benchmark TFIDF. Genetic algorithms have also been used to evolve programs that can match or beat TF-IDF schemes. (Oren, 2002) employs this method to evolve a large colony of individuals. Using the main ideas of genetic programming, mutation, crossover and copying, the author of the paper was able to evolve programs that performed slightly better than the common TF-IDF weighing scheme. Though the author felt the results were not considered significant, the paper shows that there is still interest in enhancing the simple TF-IDF scheme. Examining our data, the easiest way for us to enhance TFIDF would be to disregard case-sensitivity and equate words with their lexical derivations and synonyms. Future research might also include employing TF-IDF to performing searches in documents written in a different language than the query. Enhancing the already powerful TF-IDF algorithm would increase the success of query retrieval systems, which have quickly risen to become a key element of present global information exchange. More sophisticated representations might be able to identify the common features between the highly similar document pairs currently being missed.

# Bibliography

1. Distributed Representations of Sentences and Documents - Quoc Le, Tomas Mikolov
2. A Hybrid Document Feature Extraction Method Using Latent Dirichlet Allocation and Word2Vec - Zhibo Wang, Long Ma, and Yanqing Zhang
3. An Effective, Low-Cost Measure of Semantic Relatedness obtained from Wikipedia Links - David Milne, Ian H. Witten
4. Evaluating WordNet-based Measures of Lexical Semantic Relatedness - Alexander Budanitsky, Graeme Hirst
5. Semantic diversity: A measure of semantic ambiguity based on variability in the contextual usage of words - Paul Hoffman & Matthew A. Lambon Ralph & Timothy T. Rogers
6. Detecting Opinion Leaders and Trends in Online Social Networks - Freimut Bodendorf and Carolin Kaiser
7. Broadly speaking: Vocabulary in semantic dementia shifts towards general, semantically diverse words - Paul Hoffman, Lotte Meteyard, Karalyn Patterson
8. Visualizing the Signatures of Social Roles in Online Discussion Groups – JoSS Article: Volume 8, Howard T. Welser, Eric Gleave, Danyel Fisher and Marc Smith

# Appendix

*# -\*- coding: utf-8 -\*-*

import gensim

from gensim import corpora, models, similarities

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from datetime import datetime

import re

startTime = datetime.now()

raw\_documents = [""]

gen\_docs = [[w for w in word\_tokenize(text)] for text in raw\_documents]

*#print(gen\_docs)*

dictionary = gensim.corpora.Dictionary(gen\_docs)

corpus = [dictionary.doc2bow(gen\_doc) for gen\_doc in gen\_docs]

tf\_idf = gensim.models.TfidfModel(corpus, normalize=False)

sims = gensim.similarities.Similarity("~/Documents/bworkingdoc.txt", tf\_idf[corpus], num\_features=**len**(dictionary))

*#print(sims)*

query\_doc = [w for w in word\_tokenize("")]

query\_doc\_bow = dictionary.doc2bow(query\_doc)

*#print(" this is query doc bow", query\_doc\_bow)*

query\_doc\_tf\_idf = tf\_idf[query\_doc\_bow]

*#print(query\_doc\_tf\_idf)*

**print**(list(**enumerate**(sims[query\_doc\_tf\_idf])))

**print** datetime.now() – startTime

|  |
| --- |
|  |
|  | from flask import Flask  from flask\_restplus import Resource, Api, fields, marshal\_with |
|  | import newspaper,json |
|  | from newspaper import Article |
|  | import nltk |
|  | app = Flask(\_\_name\_\_) |
|  | api = Api(app) |
|  |  |
|  | @api.route('/api/v1/get-articles/<path:site>') |
|  | @api.doc(params={'site':"Site name with http prefix Ex:http://ndtv.com"}) |
|  | class ArticleList(Resource): |
|  | def get(self,site): |
|  | print site |
|  | paper=newspaper.build(site) |
|  | articles={} |
|  | i=0 |
|  | for article in paper.articles: |
|  | articles[i]={} |
|  | articles[i]['url']=article.url |
|  | i=i+1 |
|  | return {'size':i,'articles':articles} |
|  |  |
|  | @api.route('/api/v1/feed-url/<path:site>') |
|  | class FeedList(Resource): |
|  | def get(self,site): |
|  | paper=newspaper.build(site) |
|  | feed\_urls={} |
|  | i=0 |
|  | for feed in paper.feed\_urls(): |
|  | print feed |
|  | print type(feed) |
|  | feed\_urls[i]=feed |
|  | i=i+1 |
|  | return {'size':i,'feed\_urls':feed\_urls} |
|  |  |
|  |  |
|  | @api.route('/api/v1/scrape-article/<path:url>/<string:name>/<string:profession>') |
|  | class ArticleInfo(Resource): |
|  | def get(self,url,name,profession): |
|  | article = Article(url) |
|  | article.download() |
|  | article.parse() |
|  | article.nlp() |
|  | article\_data = {} |
|  | article\_data['url']=url |
|  | article\_data['title']=article.title |
|  | article\_data['keywords']=article.keywords |
|  | article\_data['summary']=article.summary |
|  | article\_data['text']=article.text |
|  | article\_data['top\_image']=article.top\_image |
|  | article\_data['publish\_date']=str(article.publish\_date) |
|  | article\_data['authors']=article.authors |
|  | article\_data['movies']=article.movies |
|  | article\_data['html']=article.html |
|  | ################################### |
|  | ###### ToDo: NLP checks goes here |
|  | for sent in nltk.sent\_tokenize(article.text): |
|  | for chunk in nltk.ne\_chunk(nltk.pos\_tag(nltk.word\_tokenize(sent))): |
|  | if hasattr(chunk, 'node'): |
|  | if chunk.node=="PERSON": |
|  | print chunk.node, ' '.join(c[0] for c in chunk.leaves()) |
|  | nameHit=False |
|  | profHit=False |
|  | firstNameHit=False |
|  | lastNameHit=False |
|  | nameList=name.split() |
|  | for x in article.text.split(): |
|  | if x.lower()==nameList[0].lower(): |
|  | firstNameHit=True |
|  | if x.lower()==nameList[1].lower(): |
|  | lastNameHit=True |
|  | if x.lower()==profession.lower(): |
|  | profHit=True |
|  | if firstNameHit==True and lastNameHit==True: |
|  | nameHit=True |
|  | return {'article':article\_data,'name\_hit':nameHit,'profession\_hit':profHit} |
|  | @api.route('/index') |
|  | class Home(Resource): |
|  | def get(self): |
|  | return {'hello': 'world'} |
|  |  |
|  |  |
|  | if \_\_name\_\_ == "\_\_main\_\_": |
|  | app.run(debug=True) |