Project

Document Search and Sentiment Analysis

Kieyn Parks Lavanya Rao Sarang Deotale

Abstract

In this project we aim to build a knowledge base using Natural Language Processing (NLP) and Text Retrieval approaches.The goal of this project is to build a customized ranking algorithm, which can be applied on a data set derived from twitter, to answer user queries. We aim at constructing and enhancing a structured knowledge base from natural text in a format that machines can process and use to answer human queries.

We plan to achieve this by extracting twitter feeds based on certain general group of product/services. This will serve as our initial data set, which will then be split into training data and test data. The training data will be utilized in building and improving our document ranking algorithm. Finally the test data will be used to record the performance of the algorithm.

The project may be extended by adding a sentiment analyzer. The purpose of the sentiment analyzer is to be able to automatically classify the tweets as positive, negative or neutral based on certain chosen keywords.

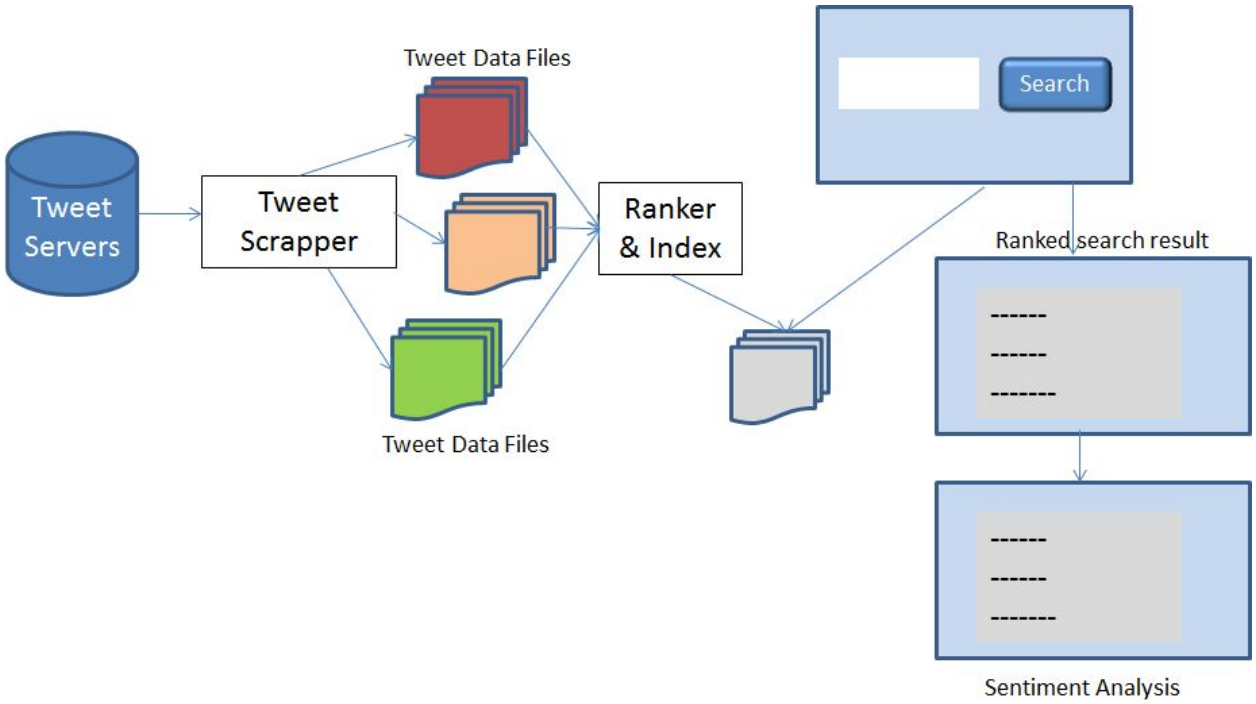
Introduction

In today's world humans are generating massive amounts of data. Through or own personal activities like shopping, travelling and social media or through the actions of machines, and sensors petabyte of information is being generated. We propose to create a text retrieval system (pull architecture) that can take a query from the user, scan a collection of document and return a ranked list of subdocuments which we can do a sentiment analysis on. We plan to aggregate Twitter tweets and form a collection of interest in 5 categories.

1) Trending news.
2) Health
3) Sports
4) Beauty
5) Technology.

This information can be used to recognize recent industry trends and how people feel about those trends.We will extract the text portion of the tweets to form a large collection of tweet documents. This collection will **theoretically** represent the entirety of all the text data available for search. We will then create an index and a ranker to score the data. Finally, if time permits, we will like to integrate '*push*' functionality into the application where users will be given recommendations for tweets of interest.

Solution Approach



High level component design

1. Tweet Scraper component –

Tweet Scraper is built using Tweepy libraries to use twitter provided APIs, first step is to create developer account on twitter, and the request authentication credentials to be able to use the API.

The authentication information received from twitter developer account is required to build the scraping application. The code opens the cursor with twitter and subscribe to various topics within search query, the returned tweets are cleaned by removing non ascii characters, removed spaces and saved in file.

2. Ranker and Index component :

The **metapy** library is being used to create the inverted index for fast search.

We used InL2, Inverse Document Frequency model with Laplace after-effect and Normalization. InL2 is a DFR-based model (Divergence from Randomness) based on the Geometric distribution and Laplace law of succession. The DFR models are based on this idea: “The more the divergence of the within-document term-frequency from its frequency within the collection, the more the information carried by the word t in the document d” . For this model, the relevance score of a document D for a query Q is given by: (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4609525/)

The retrieval function: InL2

$$Score(Q,D) = \sum_{t \in Q \cap D} c(t,Q) \cdot \frac{tfn}{tfn+c} \cdot \log_2\left(\frac{N+1}{c(t,C)+0.5}\right)$$

where $tfn = c(t,D) \cdot \log_2\left(1 + \frac{avgdl}{|D|}\right)$

It uses the following variables:

- Q, D, t : the current query, document, and term
- N: the total number of documents in the corpus C
- $avgdl$: the average document length
- $c > 0$: is a parameter

Statistical significance testing:

In this test we compared the InL2 scorer with the industry standard BM25 scorer.

First we capture the mean average precision (MAP) for each ranker and save the results to a file. Then we used R to get the statistical significance of the rankers.

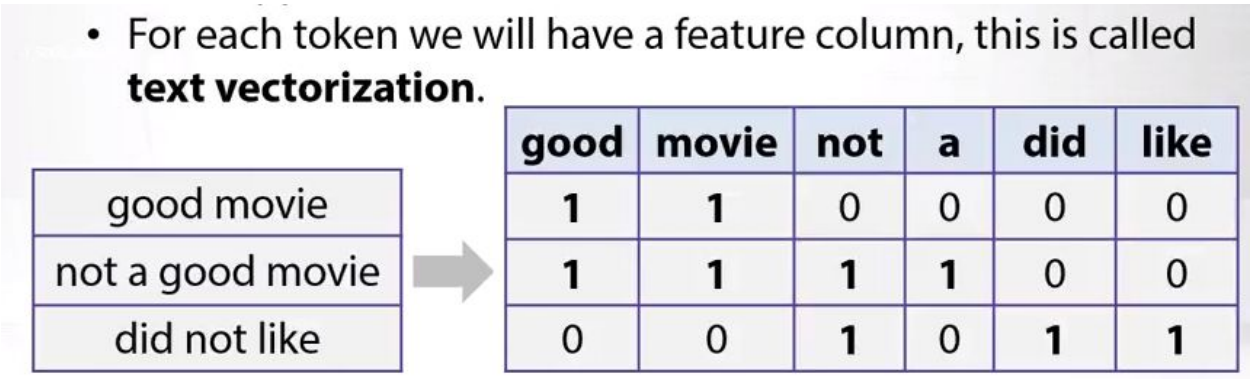
```
> inl2 = read.table('C:\\Users\\super\\Desktop\\CS-410 Text Systems\\MP2-FA19_part2\\inl2.avg_p.txt')$V1

> bm25 = read.table('C:\\Users\\super\\Desktop\\CS-410 Text Systems\\MP2-FA19_part2\\bm25.avg_p.txt')$V1
> t.test(bm25, inl2, paired=T)

        Paired t-test

data:  bm25 and inl2
t = 5.3316, df = 224, p-value = 2.373e-07
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.01436069 0.03120056
sample estimates:
mean of the differences
             0.02278063
```

The p-value is 2.373e-07 with a 95% confidence and an average difference of 0.02278.

3. Sentiment analysis:

Sentiment analysis is contextual mining of text which identifies and extract subjective information in source material, and helping a business to understand the social sentiment of their brand, product or service while monitoring online conversations.

The model is based on tokenizing words into bigrams and comparing these bag-of-words with a list of words; during this process stop words are removed from the data because they do not add any value to the analytics.



youtube.com/machine learning tv.

The remaining tokens are given a **negative** and **positive** sentiment score by comparing each word in the BOW to a prerecorded list of words that are labeled either positive or negative (lexicons). Each word is then given a value of “1”

for positive and “-1” for negative, or “0” for neutral this creates a vector representation of the document as shown below.

1) **The restuarant was great, I loved the food!**

2) **The restaurant was bad, the food was horrible!**

Positive: +1
Negative: -1
Neutral: 0

BOW:	**Restaurant,**	**great,**	**loved,**	**food**
Sentiment:	**0**	**1**	**1**	**0**

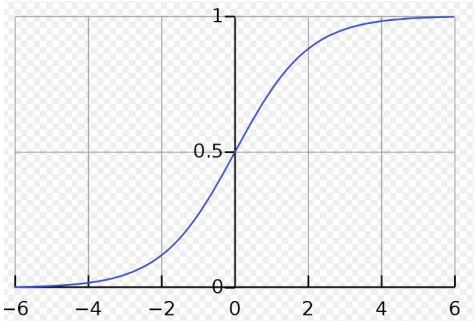
score = 0+1+1+0 = 2

BOW:	**Restaurant,**	**bad,**	**food,**	**horrible**
Sentiment:	**0**	**-1**	**0**	**-1**

score = 0+(-1)+0+(-1) = -2

The score is put through a sigmoid activation function to give a final score that is distributed between (-1,1).

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

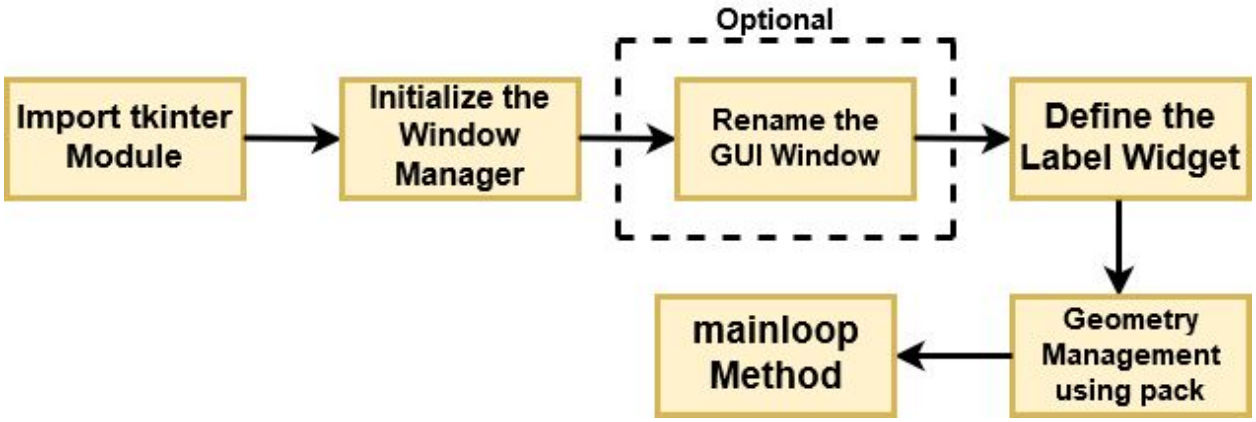


4. User Interface flow :

Tkinter is Python's de-facto standard GUI (Graphical User Interface) package. It is a thin object-oriented layer on top of Tcl/Tk. **Tkinter** is not the only GuiProgramming toolkit for Python. It is however the most commonly used one. Graphical User Interfaces with Tk, a chapter from the Python Documentation.

TkInter package is shipped with python as a standard package, so we don't need to install anything to use it.

Flow Diagram for Rendering a Basic GUI: **searchgui.py**



Let's break down the above flow diagram and understand what each component is handling!

- First, you import the key component, i.e., the Tkinter module.
- As a next step, you initialize the window manager with the tkinter.Tk() method and assign it to a variable. This method creates a blank window with close, maximize, and minimize buttons on the top as a usual GUI should have.
- Then as an optional step, you will Rename the title of the window as you like with window.title(title_of_the_window).

- Next, you make use of a widget called Label, which is used to insert some text into the window. Alternatively, you can use any of the gui objects provided by tkinter like button, entry (similar to text area) etc.

- Then, you make use of Tkinter's geometry management attribute called pack() to display the widget in size it requires. You can also specify the location for the object using the grid method, where you provide the row number and the column number where object needs to be placed. This divides the window into a grid with rows and columns and displays the object accordingly.

- Finally, as a last step, you use the mainloop() method to display the window until you manually close it. It runs an infinite loop in the backend.

Dependencies:

- pip install tkinter
- pip install textblob
- pip install tweepy

Example tweets:

Each tweet represents 1 document.

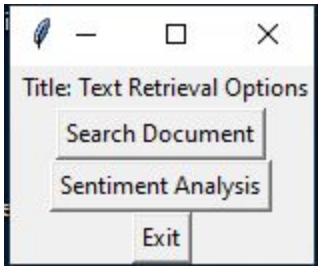
u"@SoCuteDollLora Such beauty ❤❤ love Cutest love nose, cheek, lips, chin and amazing eye any where like on Earth❤❤ Have a great rest of… https://t.co/M3AjUbpWZA"
u"RT @jaz_jenn10010: This stuff is AMAZING! It whitened my teeth after one use 😁✨I got mine at 👉 https://t.co/6PWQjg6OLN https://t.co/O6x2Q…"
u"@SoCuteDollLora Such beauty ❤❤ love Cutest love nose, cheek, lips"

Search

Example walkthrough of Search:

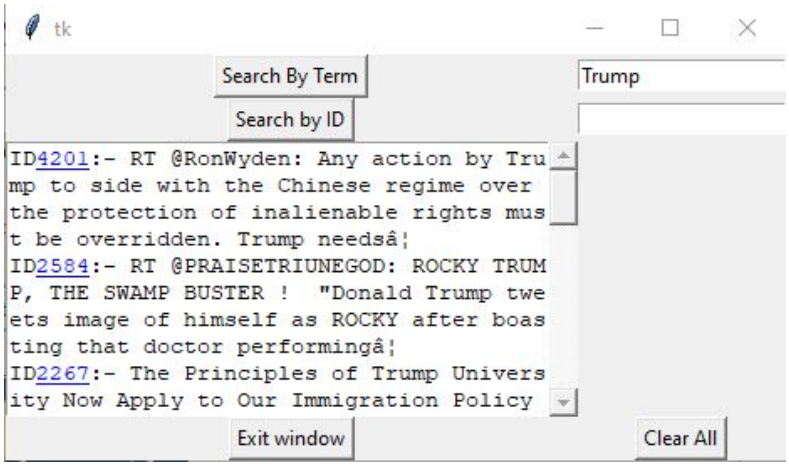
Launch : searchgui.py

Main menu:



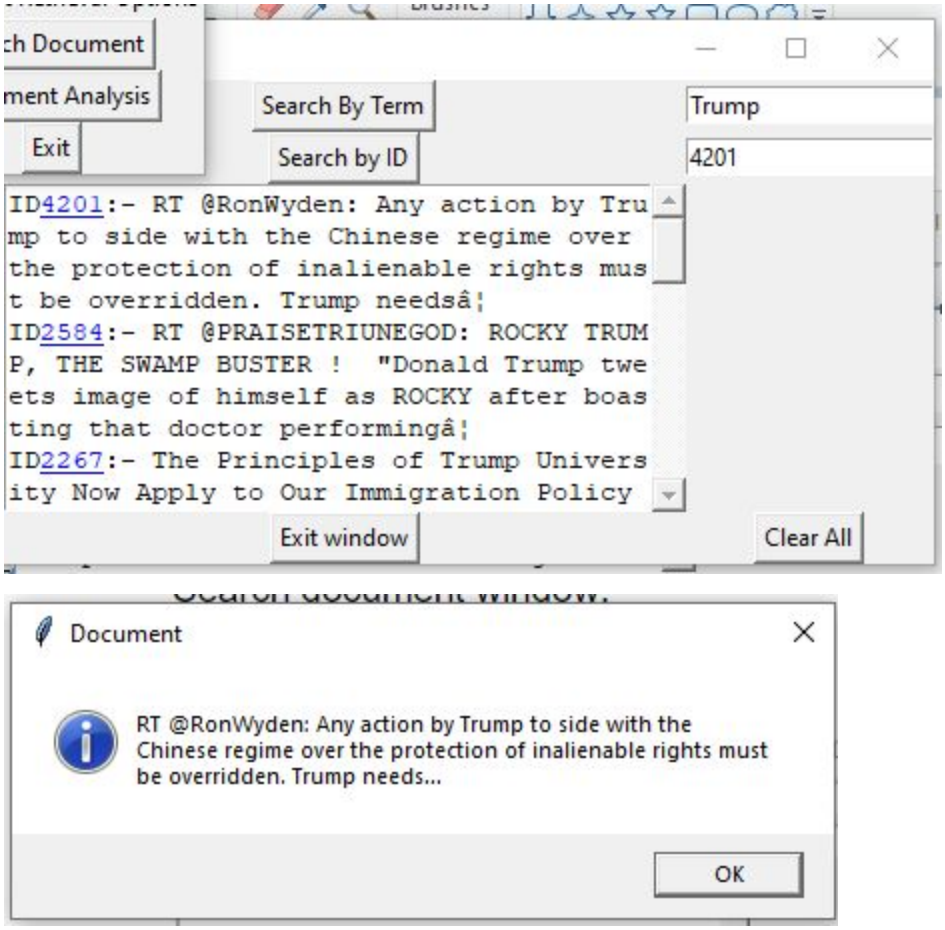
Search document window:

Search by Term: Trump.



In this window you can search by id to see the entire document in a separate window.

Search for document: 4201.



NB. Clear all will refresh the window for a new search task.

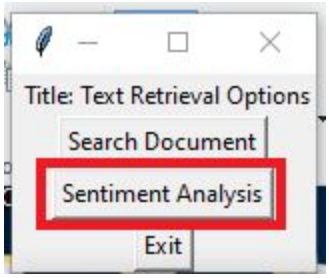
Sentiment Analysis:

Example of Simplest Sentiment Analysis:

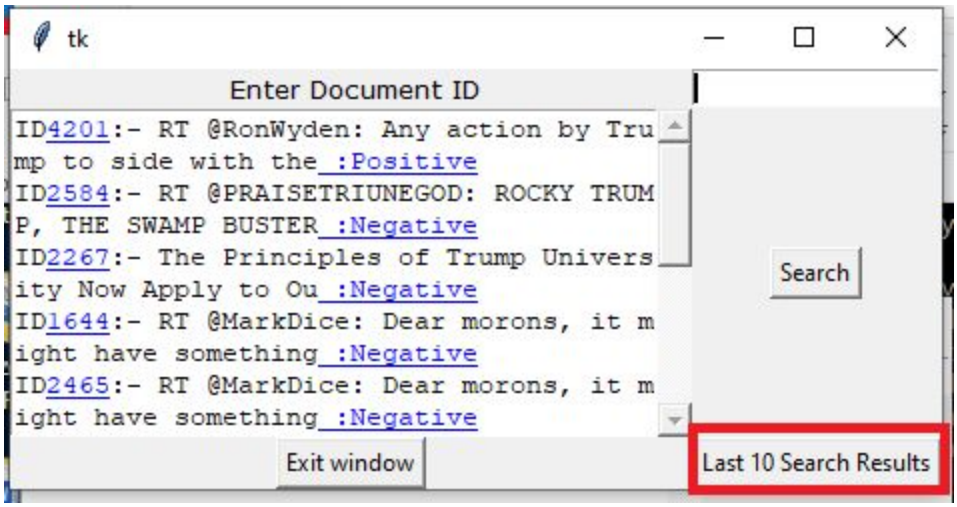
The application uses the python library “textblob” with must be installed.

For the sentiment analysis to work you must for search a Term in the search window.

From the main menu press the Sentiment Analysis button.



The Sentiment Window.



The “last 10 Search Results” button will return the last 10 search results with the Sentiment analysis results for each document. Here you can view a single document by entering the “ID” of the document.

Pros:

- simple sentiment analysis using the python library is quick and easy to implement.
- Built in lexicon.
- Built in stop words.
- Building tokenizer.

Cons:

- Cannot easily do 2-grams or more to maintain word order.
- Cannot do n-gram probability distribution.
- Not able to handle extended unicode emoji characters which can aid in sentiment analysis.

References:

- Benkoussas, CB, Bellot, PB (2015 Sep 20) Information and Graph Analysis Approaches for Book Recommendation, Retrieved from:
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4609525/
- Machine learning tv, (2018, Jul 23), Feature extraction from text, retrieved from:
- https://www.youtube.com/watch?v=7YacOe4XwhY&t=128s
- https://www.tutorialspoint.com/python/python_gui_programming.htm

Appendix A: Using the tkhyperlinkManager module

Copy the below code and save it under the python directory Python36\Lib\site-packages as tkHyperlinkManager.py

```
from tkinter import *
class HyperlinkManager:
	def __init__(self, text):
	self.text = text
	self.text.tag_config("hyper", foreground="blue", underline=1)
	self.text.tag_bind("hyper", "<Enter>", self._enter)
	self.text.tag_bind("hyper", "<Leave>", self._leave)
	self.text.tag_bind("hyper", "<Button-1>", self._click)
	self.reset()
	def reset(self):
	self.links = {}
	def add(self, action):
	# add an action to the manager.  returns tags to use in
	# associated text widget
	tag = "hyper-%d" % len(self.links)
	self.links[tag] = action
	return "hyper", tag
	def _enter(self, event):
	self.text.config(cursor="hand2")
	def _leave(self, event):
	self.text.config(cursor="")
	def _click(self, event):
	for tag in self.text.tag_names(CURRENT):
	if tag[:6] == "hyper-":
		self.links[tag]()
		return
```

Test code for EWC check.

```
from tkinter import *
window = Tk()

window.title("Welcome to the group")

window.mainloop()
```