# Packages

## Spacy installation

$ pip install spacy

* Run anaconda cmd in administrator mode

$ conda info –envs

* Choose environment (eg: C:\Users\abhijith.m\AppData\Local\conda\conda\envs\python36)
* Activate environment

$ activate C:\Users\abhijith.m\AppData\Local\conda\conda\envs\python36

* Output : (python36) C:\Windows\system32>

$ pip install <https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-2.0.0/en_core_web_sm-2.0.0.tar.gz#egg=en_core_web_sm>

or

$ python -m spacy download en\_core\_web\_sm

* and in your python file:
* import en\_core\_web\_sm
* nlp = en\_core\_web\_sm.load()

## spelling suggession

### pyenchant

* pip3 install pyenchant==1.6.6

### Hunspell

* pip install hunspell

# Text processing

## Tokenizing

### nltk

#### TweetTokenizer

from nltk.tokenize import TweetTokenizer

tknzr = TweetTokenizer()

s0 = "This is a cooool #dummysmiley: :-) :-P <3 and some arrows < > -> <--"

tknzr.tokenize(s0)

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

output - ['This', 'is', 'a', 'cooool', '#dummysmiley', ':', ':-)', ':-P', '<3', 'and', 'some', 'arrows', '<', '>', '->', '<--']

## Stopword removal

### NLTK

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

example\_sent = "This is a sample sentence, showing off the stop words filtration."

stop\_words = set(stopwords.words('english'))

word\_tokens = word\_tokenize(example\_sent)

filtered\_sentence = [w for w in word\_tokens if not w in stop\_words]

filtered\_sentence = []

for w in word\_tokens:

if w not in stop\_words:

filtered\_sentence.append(w)

print(word\_tokens)

print(filtered\_sentence)

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Output:

['This', 'is', 'a', 'sample', 'sentence', ',', 'showing',

'off', 'the', 'stop', 'words', 'filtration', '.']

['This', 'sample', 'sentence', ',', 'showing', 'stop',

'words', 'filtration', '.']

## Web scawling

### Python libaries

#### installation

pip install beautifulsoup4

pip install requests

pip install urllib3

### Read website from url

import urllib.request

with urllib.request.urlopen(url1) as response:

html = response.read()

### Beautifulsoup

from bs4 import BeautifulSoup

soup = BeautifulSoup(html)

print(soup.prettify())

#### list the children

ch = list(soup.children)

# list the type children

[type(item) for item in list(soup.children)]

* Output> [bs4.element.Doctype, bs4.element.Tag, bs4.element.NavigableString]

#### get all tags

tag = "<class 'bs4.element.Tag'>"

tags = [item for item in list(soup.children) if tag == str(type(item))]

print(tags)

#### find all – get body contents

body = tags[0].find\_all('body')

len(body)

body\_tags = list(body[0].children)

body\_tags

#### get text

body = tags[0].find\_all('body')

len(body)

p = body[0].find\_all('p')

p[0].get\_text()

#### Searching for tags by class and id

# class

soup.find\_all('p', class\_='outer-text')

soup.find\_all(class\_="outer-text")

# id

soup.find\_all(id="first")

## Stemmer

### porter

# porter stemmer

from nltk.stem.porter import \*

stemmer = PorterStemmer()

plurals = ['caresses', 'flies', 'dies', 'mules', 'denied',

'died', 'agreed', 'owned', 'humbled', 'sized',

'meeting', 'stating', 'siezing', 'itemization',

'sensational', 'traditional', 'reference', 'colonizer',

'plotted']

singles = [stemmer.stem(plural) for plural in plurals]

print(' '.join(singles)) # doctest: +NORMALIZE\_WHITESPACE

### krovetz

Installation > pip install krovetz

import krovetz

ks = krovetz.PyKrovetzStemmer()

ks.stem('walked')

## Morphological analysis

### Polyglot \*\*

#### Installation

$ pip install polyglot

* Download this files:
* <https://download.lfd.uci.edu/pythonlibs/g5apjq5m/PyICU-2.3.1-cp36-cp36m-win_amd64.whl>
* <https://download.lfd.uci.edu/pythonlibs/g5apjq5m/pycld2-0.31-cp36-cp36m-win_amd64.whl>
* If it is not working go to this site and download apropriate PyICU and pycld2 file
* <https://www.lfd.uci.edu/~gohlke/pythonlibs/>
* pip install pycld2-0.31-cp36-cp36m-win\_amd64.whl
* pip install Morfessor-2.0.4-py2.py3-none-any.whl
* git clone <https://github.com/aboSamoor/polyglot.git>
* cd polyglot
* python setup.py install

Or try using this method

* pip install polyglot

PyICU wraps the ICU (International Components for Unicode) library.

PyICU‑2.3.1‑cp27‑cp27m‑win32.whl

PyICU‑2.3.1‑cp27‑cp27m‑win\_amd64.whl

PyICU‑2.3.1‑cp35‑cp35m‑win32.whl

PyICU‑2.3.1‑cp35‑cp35m‑win\_amd64.whl

PyICU‑2.3.1‑cp36‑cp36m‑win32.whl

PyICU‑2.3.1‑cp36‑cp36m‑win\_amd64.whl

PyICU‑2.3.1‑cp37‑cp37m‑win32.whl

PyICU‑2.3.1‑cp37‑cp37m‑win\_amd64.whl

the 27 means Python 2.7 and the 36 Python 3.6... If you have 64 bits python and windows then choose the amd64 otherwhise the win32 version.

* polyglot download embeddings2.en
* polyglot download ner2.en

#### Morphological Analysis

#### Download Necessary Models

* polyglot download morph2.en

#### Example

<https://polyglot.readthedocs.io/en/latest/MorphologicalAnalysis.html>

from polyglot.text import Text, Word

blob = "Wewillmeettoday."

text = Text(blob)

text.language = "en"

text.morphemes

* WordList([u'We', u'will', u'meet', u'to', u'day', u'.'])

## Named Entity Recognition \*\*\*\*\*

Using spaCy

<https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da>

## Spelling Correction

### phonetics

import re

p = re.compile('(u|e|i|o|u|y|h|w)')

n1 = re.compile('(b|f|p|v)')

n2 = re.compile('(c|g|j|k|q|s|x|z)')

n3 = re.compile('(d|t)')

n4 = re.compile('(l)')

n5 = re.compile('(m|n)')

n6 = re.compile('(r)')

dh = re.compile('(-)')

# rd = re.compile(r'([0-9])(.+)\1')

class Spelling:

def phonetic(self, word):

word = word.lower()

# keep first letter upper

word = word[0].upper() + word[1:]

# Replace letter with hyphens u, e, i, o, u, y, h, w

word = p.sub("-",word)

# replace other letter with numbers

word = n1.sub("1",word)

word = n2.sub("2",word)

word = n3.sub("3",word)

word = n4.sub("4",word)

word = n5.sub("5",word)

word = n6.sub("6",word)

word = dh.sub("",word)

while re.search(r'([0-9])(.\*)\1', word):

word= re.sub(r'([0-9])(.\*)\1', r'\1\2', word)

return word

word1 = "poiner"

word2 = "pointer"

print(objSpelling.phonetic(word1))

print(objSpelling.phonetic(word2))

# document similarity

## Gensim

### Prerequisites[¶](https://radimrehurek.com/gensim/simserver.html#prerequisites)

* pip install simserver

OR

* git clone <https://github.com/RaRe-Technologies/gensim-simserver.git>
* cd simserver

**>>> from** **simserver** **import** SessionServer

**>>>** server = SessionServer('/tmp/my\_server') *# resume server (or create a new one)*

It is assumed you have gensim properly [**installed**](https://radimrehurek.com/gensim/install.html). You’ll also need the [**sqlitedict**](https://pypi.python.org/pypi/sqlitedict) package that wraps Python’s sqlite3 module in a thread-safe manner:

* pip install sqlitedict

To test the remote server capabilities, install Pyro4 (Python Remote Objects, at version 4.8 as of this writing):

$ sudo easy\_install Pyro4

* pip install Pyro4

Don’t forget to initialize logging to see logging messages:

**>>> import** **logging**

**>>>** logging.basicConfig(format='*%(asctime)s* : *%(levelname)s* : *%(message)s*', level=logging.INFO)

# Logic Programming

## **Kanren**

* **Kanren-** It lets us express logic as rules and facts and simplifies making code for business logic.
* pip install kanren
* **SymPy-** This is a Python library for symbolic mathematics. It is nearly a full-featured Computer Algebra System.
* pip install sympy

# Index creation

## {“word” : (doc\_num,count,[position])} – (1,2,[5,10]) – 1,2,5,10

from nltk.tokenize import word\_tokenize

import os

import pickle

from nltk.stem.porter import \*

# read corpus. The data folder contain Wikipedia pages of different topics

path = "data/"

word\_files = os.listdir(path)

word\_set = set()

word\_dict = {}

# index creation

class IndexCreation:

def \_\_init\_\_(self):

# wordset creation

self.word\_set\_creation()

self.word\_dict\_creation()

def add\_new\_w\_index(self,w, doc\_name, pos\_word):

pdb.set\_trace()

# word\_dict[w]

next\_doc\_index = 0

while True:

try:

cur\_doc = word\_dict[w][next\_doc\_index]

is\_no\_error = True

except:

word\_dict[w].append(doc\_name)

word\_dict[w].append(1)

word\_dict[w].append(pos\_word)

update = True

is\_no\_error = False

if is\_no\_error:

if cur\_doc == doc\_name:

count = word\_dict[w][next\_doc\_index+1]

# update count with one value

word\_dict[w][next\_doc\_index+1] = count + 1

# add position

word\_dict[w].append(pos\_word)

update = True

else:

count = word\_dict[w][next\_doc\_index+1]

next\_doc\_index = next\_doc\_index + 1 + count + 1

update = False

if update :

break

def create\_index(self, doc\_name, doc\_list):

pos\_word = 0

for sent in doc\_list:

for w in sent:

pos\_word += 1

self.add\_new\_w\_index(w,doc\_name, pos\_word)

def word\_set\_creation(self):

# doc\_to\_token

for file\_name in word\_files:

doc\_list = self.doc\_to\_token(file\_name)

# print(doc\_list)

# tokenization

for line in doc\_list:

for word in line:

word\_set.add(word)

def doc\_to\_token(self, file\_name):

with open(path + file\_name, encoding='utf-8') as f:

doc\_list = []

for line in f:

# line = line.decode('utf-8')

doc\_list.append(word\_tokenize(line))

return doc\_list

def word\_dict\_creation(self):

s\_word\_set = sorted(word\_set)

for w in s\_word\_set:

word\_dict[w] = []

index = IndexCreation()

# individual index creation

# doc\_list = index.doc\_to\_token("Bank.txt")

# index.create\_index(0,doc\_list)

file\_count = 0

for filename in word\_files:

print("Index creation of %d -> %s" % (file\_count,filename))

doc\_list = index.doc\_to\_token(file\_name=filename)

index.create\_index(file\_count,doc\_list)

file\_count+=1

file = open('index.pickle', 'wb')

pickle.dump(word\_dict, file)

file.close()

# Clustering

## Diece coeficent

It is the continuation of [index creation](word#_{)

file = open('index.pickle', 'rb')

data = pickle.load(file)

file.close()

print(data)

# create a cluster

# first 1000 words of dictionary

word\_dict\_1000 = {}

count = 0

for w in data:

if count > 700:

word\_dict\_1000[w] = data[w]

if count == 1700:

break

count += 1

# display some words

count = 0

for w in word\_dict\_1000:

if count==1000:

break

count += 1

if count > 700:

print(count, w,word\_dict\_1000[w])

# create stem words using porter stemmer

stemmer = PorterStemmer()

stem\_class = {}

print(stemmer.stem("Active") )

for w in word\_dict\_1000:

stem\_w = stemmer.stem(w)

print(stem\_w,w)

try:

stem\_class[stem\_w].append(w)

print("\*" \* 20,stem\_class[stem\_w])

except:

stem\_class[stem\_w] = [w]

print("-" \* 20,stem\_class[stem\_w])

# Dice coefficent at document level

# - na number of windows containing word a

# - nb number of windows containing word b

# - nab number of windows contianing bath a & b

# - window size is between 50 - 100

class Diece:

def \_\_init\_\_(self,window=80):

self.window = window

def nw(self,word):

na = {}

word\_pos\_index = word\_dict\_1000[word]

len\_index = len(word\_pos\_index)

next\_word\_pos = 0

while (len\_index > next\_word\_pos):

doc = word\_pos\_index[next\_word\_pos]

count = word\_pos\_index[next\_word\_pos + 1]

# calculating na

na[doc] = count \* self.window

# for p in range(count):

# print("pos",p,word\_pos\_index[next\_word\_pos + 1+p+1])

next\_word\_pos = next\_word\_pos + 1 + count + 1

return na

def nab(self,word\_a, word\_b):

# set\_trace()

nab = {}

word\_a\_pos\_index = word\_dict\_1000[word\_a]

word\_b\_pos\_index = word\_dict\_1000[word\_b]

len\_a\_index = len(word\_a\_pos\_index)

len\_b\_index = len(word\_b\_pos\_index)

next\_a\_word\_pos = next\_b\_word\_pos = 0

while (len\_a\_index > next\_a\_word\_pos and len\_b\_index > next\_b\_word\_pos):

doc\_a = word\_a\_pos\_index[next\_a\_word\_pos]

doc\_b = word\_b\_pos\_index[next\_b\_word\_pos]

count\_a = word\_a\_pos\_index[next\_a\_word\_pos + 1]

count\_b = word\_b\_pos\_index[next\_b\_word\_pos + 1]

if doc\_a == doc\_b:

for a\_p in range(count\_a):

for b\_p in range(count\_b):

# calculating nab

p\_b\_p = word\_b\_pos\_index[next\_b\_word\_pos + 1 + b\_p + 1]

p\_a\_p = word\_a\_pos\_index[next\_a\_word\_pos + 1 + a\_p + 1]

d = abs(p\_b\_p - p\_a\_p)

t = self.window - d

if t > 0:

nab[word\_a + "\_" + word\_b] = t

next\_a\_word\_pos = next\_a\_word\_pos + 1 + count\_a + 1

elif doc\_a < doc\_b:

next\_a\_word\_pos = next\_a\_word\_pos + 1 + count\_a + 1

else:

next\_b\_word\_pos = next\_b\_word\_pos + 1 + count\_b + 1

return nab

obj\_Diece = Diece(window=100)

word\_a = 'Computer'

word\_b = 'Computers'

na = obj\_Diece.nw(word\_a)

nb = obj\_Diece.nw(word\_b)

nab = obj\_Diece.nab(word\_a=word\_a, word\_b=word\_b)

cooccurence = nab/(na + nb)

# Keyword extraction

## Graph based

### Text Rank

import nltk

from nltk import word\_tokenize

import string

from nltk.stem import WordNetLemmatizer

import numpy as np

import math

wordnet\_lemmatizer = WordNetLemmatizer()

adjective\_tags = ['JJ','JJR','JJS']

wanted\_POS = ['NN','NNS','NNP','NNPS','JJ','JJR','JJS','VBG','FW']

window\_size = 3

#nltk.download('punkt')

class Preprocess:

def clean(self, text):

# set\_trace()

text = text.lower()

# printable = set(string.printable)

# text = filter(lambda x: x in printable, text) #filter funny characters, if any.

return text

def tokenize(self, text):

return word\_tokenize(Cleaned\_text)

def pos\_tag(self, text):

return nltk.pos\_tag(text)

def lemma(self, text):

lemmatized\_text = []

for word in POS\_tag:

if word[1] in adjective\_tags:

lemmatized\_text.append(str(wordnet\_lemmatizer.lemmatize(word[0],pos="a")))

else:

lemmatized\_text.append(str(wordnet\_lemmatizer.lemmatize(word[0])))

return lemmatized\_text

def pos\_filter(self, POS\_tag):

stopwords = []

for word in POS\_tag:

if word[1] not in wanted\_POS:

stopwords.append(word[0])

punctuations = list(str(string.punctuation))

stopwords = stopwords + punctuations

return stopwords

def complete\_stopword\_gen(self, stopwords):

stopword\_file = open("long\_stopwords.txt", "r")

#Source = https://www.ranks.nl/stopwords

lots\_of\_stopwords = []

for line in stopword\_file.readlines():

lots\_of\_stopwords.append(str(line.strip()))

stopwords\_plus = []

stopwords\_plus = stopwords + lots\_of\_stopwords

stopwords\_plus = set(stopwords\_plus)

return stopwords\_plus

def rem\_stop\_w\_lemma(self, stopwords\_plus, lemmatized\_text):

processed\_text = []

for word in lemmatized\_text:

if word not in stopwords\_plus:

processed\_text.append(word)

return processed\_text

def vocabulary(self, processed\_text):

return list(set(processed\_text))

class Graph:

def \_\_init\_\_(self, vocabulary):

self.vocab\_len = len(vocabulary)

self.weighted\_edge = np.zeros((self.vocab\_len,self.vocab\_len),dtype=np.float32)

self.score = np.zeros((self.vocab\_len),dtype=np.float32)

self.window\_size = 3

def build\_graph(self,processed\_text, vocabulary):

covered\_coocurrences = []

for i in range(0,self.vocab\_len):

self.score[i]=1

for j in range(0,self.vocab\_len):

if j==i:

self.weighted\_edge[i][j]=0

else:

for window\_start in range(0,(len(processed\_text)-self.window\_size)):

window\_end = window\_start+self.window\_size

window = processed\_text[window\_start:window\_end]

if (vocabulary[i] in window) and (vocabulary[j] in window):

index\_of\_i = window\_start + window.index(vocabulary[i])

index\_of\_j = window\_start + window.index(vocabulary[j])

# index\_of\_x is the absolute position of the xth term in the window

# (counting from 0)

# in the processed\_text

if [index\_of\_i,index\_of\_j] not in covered\_coocurrences:

self.weighted\_edge[i][j]+=1/math.fabs(index\_of\_i-index\_of\_j)

covered\_coocurrences.append([index\_of\_i,index\_of\_j])

return covered\_coocurrences

def calculate\_waighted\_sum(self):

inout = np.zeros((self.vocab\_len),dtype=np.float32)

for i in range(0,self.vocab\_len):

for j in range(0,self.vocab\_len):

inout[i]+=self.weighted\_edge[i][j]

return inout

def scoring\_verices(self, inout):

MAX\_ITERATIONS = 50

d=0.85

threshold = 0.0001 #convergence threshold

for iter in range(0,MAX\_ITERATIONS):

prev\_score = np.copy(self.score)

for i in range(0,self.vocab\_len):

summation = 0

for j in range(0,self.vocab\_len):

if self.weighted\_edge[i][j] != 0:

summation += (self.weighted\_edge[i][j]/inout[j])\*self.score[j]

self.score[i] = (1-d) + d\*(summation)

if np.sum(np.fabs(prev\_score-self.score)) <= threshold: #convergence condition

# print ("Converging at iteration "+str(iter)+"....")

break

def phrase\_partitioning(self, lemmatized\_text, stopwords\_plus):

phrases = []

phrase = " "

for word in lemmatized\_text:

if word in stopwords\_plus:

if phrase!= " ":

phrases.append(str(phrase).strip().split())

phrase = " "

elif word not in stopwords\_plus:

phrase+=str(word)

phrase+=" "

return phrases

def create\_unique\_phrase(self,phrases):

unique\_phrases = []

for phrase in phrases:

if phrase not in unique\_phrases:

unique\_phrases.append(phrase)

return unique\_phrases

def thinned\_uniq\_phrase(self, unique\_phrases, vocabulary):

for word in vocabulary:

#print word

for phrase in unique\_phrases:

if (word in phrase) and ([word] in unique\_phrases) and (len(phrase)>1):

#if len(phrase)>1 then the current phrase is multi-worded.

#if the word in vocabulary is present in unique\_phrases as a single-word-phrase

# and at the same time present as a word within a multi-worded phrase,

# then I will remove the single-word-phrase from the list.

unique\_phrases.remove([word])

return unique\_phrases

def scoring\_keyphrase(self, unique\_phrases, vocabulary):

phrase\_scores = []

keywords = []

for phrase in unique\_phrases:

phrase\_score=0

keyword = ''

for word in phrase:

keyword += str(word)

keyword += " "

phrase\_score+=self.score[vocabulary.index(word)]

phrase\_scores.append(phrase\_score)

keywords.append(keyword.strip())

return keywords, phrase\_scores

# i=0

# for keyword in keywords:

# print ("Keyword: '"+str(keyword)+"', Score: "+str(phrase\_scores[i]))

# i+=1

def ranking\_keyphrase(self, phrase\_scores, keywords,keywords\_num = 1):

sorted\_index = np.flip(np.argsort(phrase\_scores),0)

# keywords\_num = 4

sorted\_keywords = []

# print ("Keywords:\n")

for i in range(0,keywords\_num):

sorted\_keywords.append(str(keywords[sorted\_index[i]]))

# print (str(keywords[sorted\_index[i]])+", ",)

return sorted\_keywords

object usage

Text = "SCE stated re-circuit is a game changer in utilizing dabigatran uninterrupted"

# preprocessing text

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

objPreprocess = Preprocess()

Cleaned\_text = objPreprocess.clean(text=Text)

text = objPreprocess.tokenize(Cleaned\_text)

POS\_tag = objPreprocess.pos\_tag(text)

lemmatized\_text = objPreprocess.lemma(POS\_tag)

POS\_tag = objPreprocess.pos\_tag(lemmatized\_text)

stopwords = objPreprocess.pos\_filter(POS\_tag=POS\_tag)

stopwords\_plus = objPreprocess.complete\_stopword\_gen(stopwords=stopwords)

processed\_text = objPreprocess.rem\_stop\_w\_lemma(lemmatized\_text=lemmatized\_text,

stopwords\_plus=stopwords)

vocabulary = objPreprocess.vocabulary(processed\_text=processed\_text)

# Graph creation

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

objGraph = Graph(vocabulary=vocabulary)

covered\_coocurrences = objGraph.build\_graph(vocabulary=vocabulary,

processed\_text=processed\_text)

inout = objGraph.calculate\_waighted\_sum()

objGraph.scoring\_verices(inout=inout)

phrases = objGraph.phrase\_partitioning(lemmatized\_text=lemmatized\_text, stopwords\_plus=stopwords\_plus)

unique\_phrases = objGraph.create\_unique\_phrase(phrases=phrases)

unique\_phrases = objGraph.thinned\_uniq\_phrase(vocabulary=vocabulary,

unique\_phrases=unique\_phrases)

keywords, phrase\_scores = objGraph.scoring\_keyphrase(unique\_phrases=unique\_phrases,

vocabulary=vocabulary)

final\_keywords = objGraph.ranking\_keyphrase(phrase\_scores=phrase\_scores,

keywords=keywords,

keywords\_num=4)

print(final\_keywords)

# Document similarity

## TF-IDF

### corpus

import logging

logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)

from gensim import utils

documents = ["Human machine interface for lab abc computer applications",

"A survey of user opinion of computer system response time",

"The EPS user interface management system",

"System and human system engineering testing of EPS",

"Relation of user perceived response time to error measurement",

"The generation of random binary unordered trees",

"The intersection graph of paths in trees",

"Graph minors IV Widths of trees and well quasi ordering",

"Graph minors A survey"]

### basic preprocessing

from pprint import pprint # pretty-printer

from collections import defaultdict

from gensim import models

# remove common words and tokenize

stoplist = set('for a of the and to in'.split())

texts = [

[word for word in document.lower().split() if word not in stoplist]

for document in documents

]

# remove words that appear only once

frequency = defaultdict(int)

for text in texts:

for token in text:

frequency[token] += 1

texts = [

[token for token in text if frequency[token] > 1]

for text in texts

]

pprint(texts)

### Create and store dictionary

from gensim import corpora

dictionary = corpora.Dictionary(texts)

dictionary.save('deerwester.dict') # store the dictionary, for future reference

print(dictionary)

print(dictionary.token2id)

### BOW creation

corpus = [dictionary.doc2bow(text) for text in texts]

corpora.MmCorpus.serialize('deerwester.mm', corpus) # store to disk, for later use

pprint(corpus)

### Adding new word to BOW

new\_doc = "Human computer interaction"

new\_vec = dictionary.doc2bow(new\_doc.lower().split())

print(new\_vec)

### Transformation interface

tfidf = models.TfidfModel(corpus) # step 1 -- initialize a model

corpus\_tfidf = tfidf[corpus]

for doc in corpus\_tfidf:

print(doc)

### load dict & corpus

from gensim import corpora

dictionary = corpora.Dictionary.load('deerwester.dict')

corpus = corpora.MmCorpus('deerwester.mm') # comes from the first tutorial, "From strings to vectors"

print(corpus)

### index creation & loading

from gensim import similarities

index = similarities.MatrixSimilarity(lsi[corpus])a # transform corpus to LSI space and index it

index.save('deerwester.index')

index = similarities.MatrixSimilarity.load('deerwester.index')

### calculate similarity

sims = index[vec\_lsi] # perform a similarity query against the corpus

print(list(enumerate(sims))) # print (document\_number, document\_similarity) 2-tuples

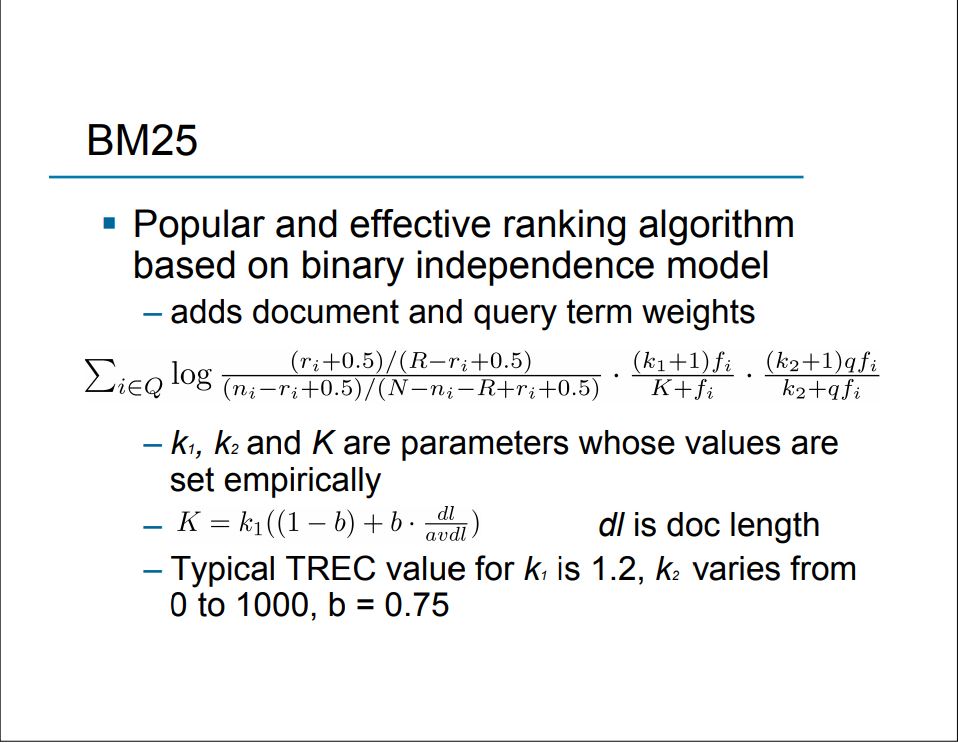
### Sort similarity

sims = sorted(enumerate(sims), key=lambda item: -item[1])

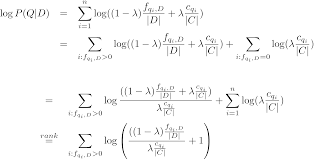
print(sims) # print sorted (document number, similarity score) 2-tuples

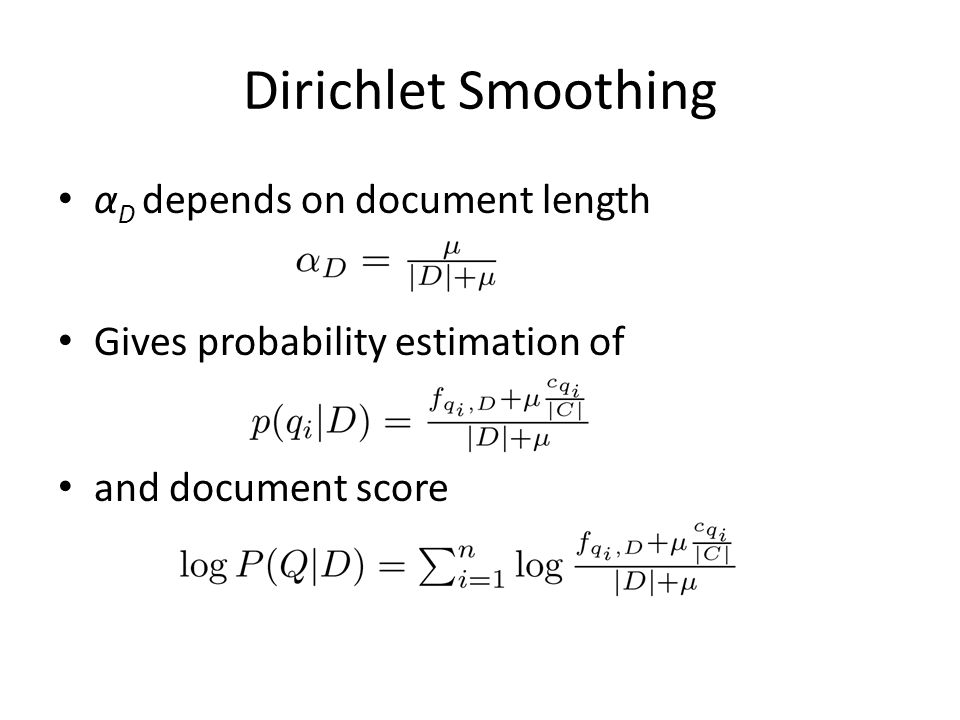
# Retrieval Models

## BM25



## Query likelihood model





import numpy as np

query = ['president', 'lincoln']

cq = [160000, 2400]

l\_d = 1800

l\_c = 10\*\*9

mu = 2000

class QL:

    def \_\_init\_\_(self, l\_d, l\_c, mu):

        self.l\_d = l\_d

        self.l\_c = l\_c

        self.mu = mu

    def doc\_score(self, fq, cq):

        score = 0

        for i in range(len(fq)):

            num = fq[i] + self.mu \* (cq[i] / self.l\_c)

            den = self.l\_d + self.mu

            score += np.log(num / den)

        return score

obj\_QL = QL(l\_d, l\_c, mu)

fq = [15, 25]

print(obj\_QL.doc\_score(fq, cq))

# output: -10.53

fq = [15, 1]

print(obj\_QL.doc\_score(fq, cq))

# output: -13.75

fq = [15, 0]

print(obj\_QL.doc\_score(fq, cq))

# output: -19.09

fq = [1, 25]

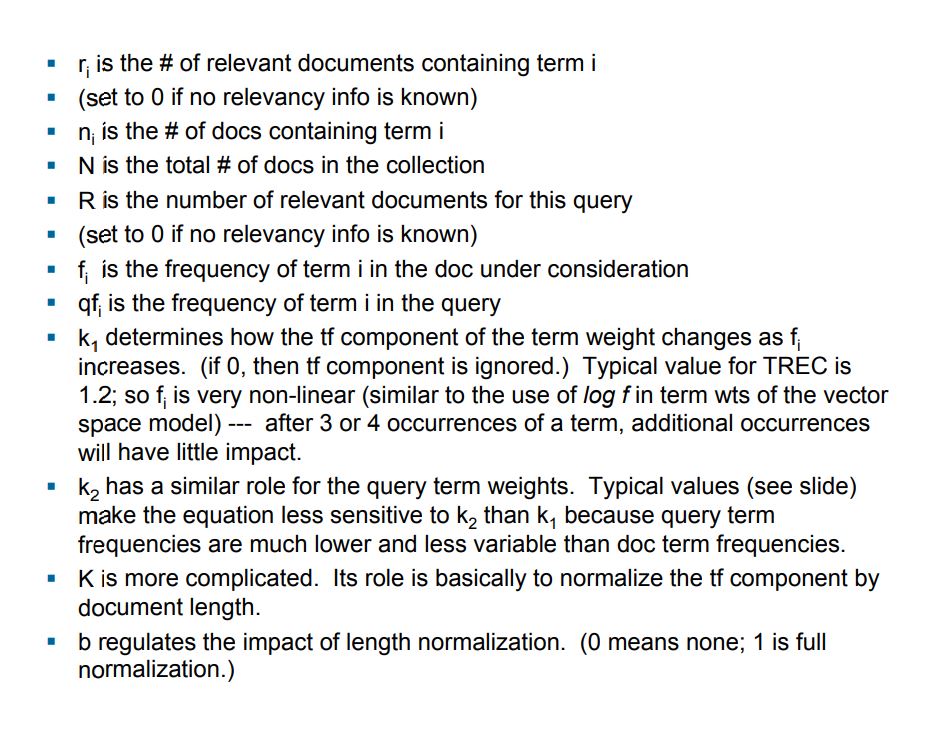
print(obj\_QL.doc\_score(fq, cq))

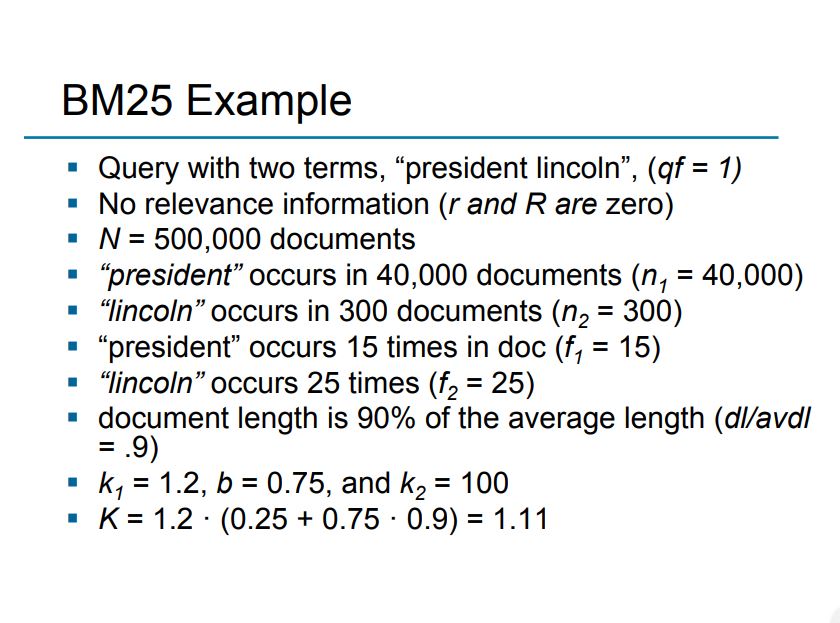
# output: -12.98

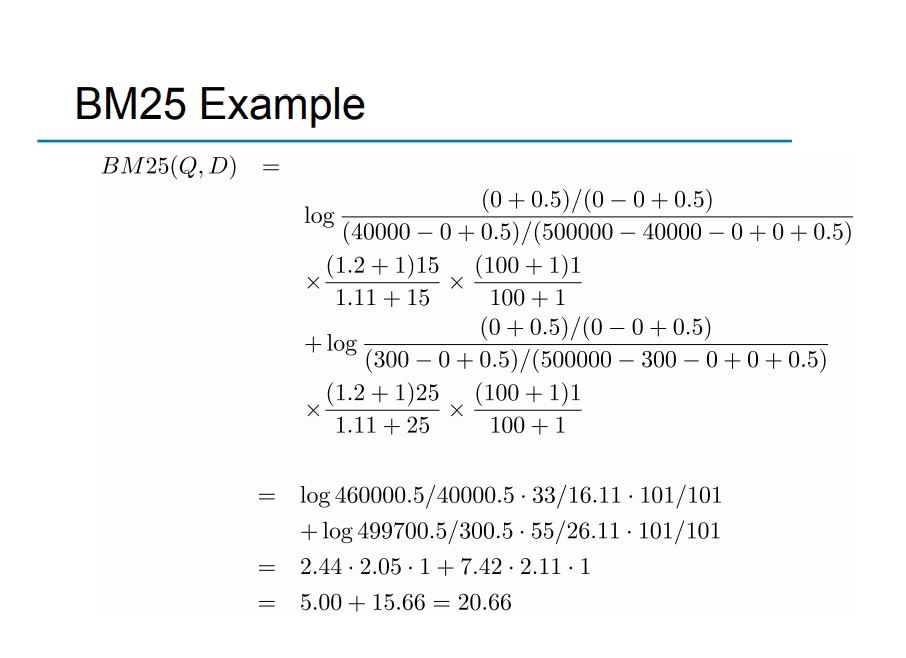
fq = [0, 25]

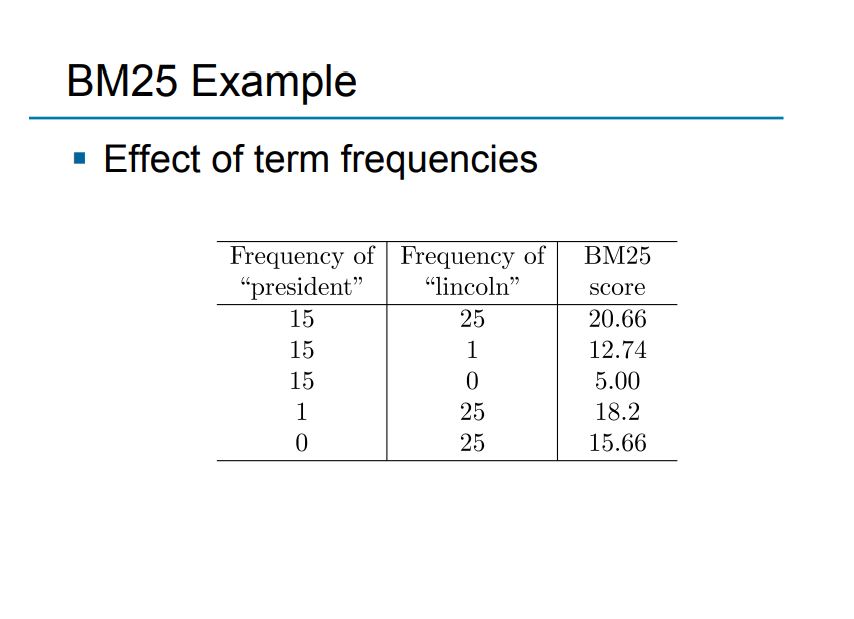
print(obj\_QL.doc\_score(fq, cq))

# output: -14.40









import numpy as np

R = 0

N = 500000

k1 = 1.2

K = 1.11

k2 = 100

class BM25:

    def \_\_init\_\_(self, R, N, k1, K, k2):

        self.R = R

        self.N = N

        self.k1 = k1

        self.K = K

        self.k2 = k2

    def PIdf(self, ri,ni):

        num = (ri + 0.5) / (self.R - ri + 0.5)

        den = (ni - ri +0.5) / (self.N - ni - self.R + ri + 0.5)

        return np.log(num / den)

    def Normalizing\_f(self, fi, qfi):

        t1 = ((self.k1 + 1) \* fi) / (self.K + fi)

        t2 = ((self.k2 + 1) \* qfi) / (self.k2 + qfi)

        return t1 \* t2

    def Bm25(self, ri, ni, fi, qfi):

        pidf = self.PIdf(ri, ni)

        norm\_f = self.Normalizing\_f(fi, qfi)

        return pidf \* norm\_f

bm25\_obj = BM25(R, N, k1, K, k2)

q = ["president", "lincoln"]

# relevance

r = [0, 0]

# no. of docs contain queries

n = [40000, 300]

# freequency of query in doc

f = [15, 25]

# query freequency

qf = [1, 1]

bm\_sum = 0

for i in range(len(q)):

    bm\_sum += bm25\_obj.Bm25(r[i], n[i], f[i], qf[i])

print(bm\_sum)

output

20.625189631270004

## BM25 pypi: <https://pypi.org/project/rank-bm25/>

# Chat bot

## Memory Network

<https://www.youtube.com/watch?v=BN7Kp0JD04o>

1. github code: <https://github.com/domluna/memn2n>

git clone git@github.com:domluna/memn2n.git

mkdir ./memn2n/data/

cd ./memn2n/data/

wget http://www.thespermwhale.com/jaseweston/babi/tasks\_1-20\_v1-2.tar.gz

tar xzvf ./tasks\_1-20\_v1-2.tar.gz

cd ../

python single.py

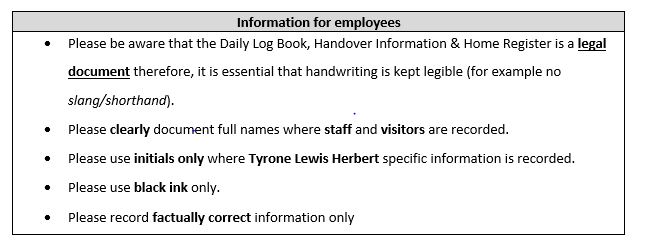
1. github code: <https://github.com/carpedm20/MemN2N-tensorflow>

* pip install future
* pip install progress
* To train a model with 6 hops and memory size of 100, run the following command:
* python main.py --nhop 6 --mem\_size 100
* python main.py –help
* (Optional) If you want to see a progress bar, install progress with pip:
* python main.py --nhop 6 --mem\_size 100 --show True
* After training is finished, you can test and validate with:
* python main.py --is\_test True --show True

1. code: <https://appliedmachinelearning.blog/2019/05/02/building-end-to-end-memory-network-for-question-answering-system-on-babi-facebook-data-set-python-keras-part-2/>

# OCR

## pytesseract



import cv2

try:

from PIL import Image

except ImportError:

import Image

import pytesseract

pytesseract.pytesseract.tesseract\_cmd = r'C:\Users\abhijith.m\AppData\Local\Tesseract-OCR\tesseract.exe'

def ocr\_core(filename):

"""

This function will handle the core OCR processing of images.

"""

img = Image.open(filename)

# print(img)

img.show()

text = pytesseract.image\_to\_string(img) # We'll use Pillow's Image class to open the image and pytesseract to detect the string in the image

return text

print(ocr\_core('table2.JPG'))

# Spell check

## pyspellchecker

<https://pypi.org/project/pyspellchecker/>

# Reference

## Sentiment analyisis

### Word2Vec + LSTM

<https://towardsdatascience.com/machine-learning-word-embedding-sentiment-classification-using-keras-b83c28087456>

# Document Classification – Doc2Vec

<https://github.com/RaRe-Technologies/movie-plots-by-genre/blob/master/ipynb_with_output/Document%20classification%20with%20word%20embeddings%20tutorial%20-%20with%20output.ipynb>

## Word2vec

<https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568>

<https://towardsdatascience.com/multi-class-text-classification-with-doc2vec-logistic-regression-9da9947b43f4>

# Auto-complete

Auto complete in [Django](https://django-haystack.readthedocs.io/en/v2.4.1/autocomplete.html)