# 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

* pip install https://github.com/explosion/spacy-models/releases/download/en\_core\_web\_sm-3.0.0/en\_core\_web\_sm-3.0.0.tar.gz
* 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))

## Train and test split pandas dataframe

from sklearn.model\_selection import train\_test\_split

    df = pd.read\_csv('data/TopicModellingData.csv', encoding='cp1252')

    train, valid, y\_train, y\_valid = train\_test\_split(

            df[['Article Title', 'Abstract Text', 'SAT ID']], df['Tag'], test\_size=0.33, random\_state=42)

    print(train, valid, y\_train, y\_valid)

# 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

# Inverted 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()

# Inverted Index Django

## Views.py

from django.shortcuts import render

from django.views.generic import View

from EmployeeApp.utils import is\_json

from EmployeeApp.mixins import HttpResponseMixin

from django.views.decorators.csrf import csrf\_exempt

from django.utils.decorators import method\_decorator

import json

from SearchApp import InvertedIndex

# urls: /search/add\_inverted\_index/

@method\_decorator(csrf\_exempt, name='dispatch')

class AddInvertedIndexView(HttpResponseMixin, View):

    def get(self, request, \*args, \*\*kwargs):

        result = {"is\_get\_method" : True}

        error = []

        data = {

            "doc\_id" : 1,

            "doc" : "is test doc"

        }

        result, error = InvertedIndex.II().save\_doc(data)

        context = json.dumps({'result' : result, 'error' : error})

        return self.render\_to\_http\_response(json\_data= context )

    def post(self, request, \*args, \*\*kwargs):

        result = {"is\_post\_method" : True}

        error = []

        context = json.dumps({'result' : result, 'error' : error})

        return self.render\_to\_http\_response(json\_data= context )

# urls: /search/ii\_search/

@method\_decorator(csrf\_exempt, name='dispatch')

class SearchInvertedIndexView(HttpResponseMixin, View):

    def get(self, request, \*args, \*\*kwargs):

        result = {"is\_get\_method" : True}

        error = []

        data = {

            "query" : "name - brand abhi"

        }

        res = InvertedIndex.II().search(data)

        result["docs"] = res

        context = json.dumps({'result' : result, 'error' : error})

        return self.render\_to\_http\_response(json\_data= context )

    def post(self, request, \*args, \*\*kwargs):

        result = {}

        error = []

        data = request.POST

        result = InvertedIndex.II().search(data)

        # result["products"] = res

        context = json.dumps({'result' : result, 'error' : error})

        return self.render\_to\_http\_response(json\_data= context )

## urls.py

 path('add\_inverted\_index/', views.AddInvertedIndexView.as\_view(), name="add\_inverted\_index"),

    path('ii\_search/', views.SearchInvertedIndexView.as\_view(), name="ii\_search"),

## InvertedIndex.py

import pdb

from nltk.tokenize import TweetTokenizer

tknzr = TweetTokenizer()

from SearchApp import db\_handler

from ProductsApp import db\_handler as prod\_db

import contractions

import spacy

from SearchApp import BestMatch25

sp = spacy.load('en\_core\_web\_sm')

# II = InvertedIndex

class II:

    def delete\_doc\_id(self, doc\_id):

        m\_prods = db\_handler.Get().all\_prod\_by\_prod\_id(doc\_id)

        for ii\_prod in m\_prods:

            # delete doc\_id

            if ii\_prod.previous\_doc:

                if ii\_prod.next\_doc:

                    data\_prod\_prev = {

                        "next\_doc" : ii\_prod.next\_doc

                    }

                    db\_handler.Update().ii\_prod(data\_prod\_prev, ii\_prod.previous\_doc)

                else:

                    data\_prod\_prev = {

                        "next\_doc" : None

                    }

                    db\_handler.Update().ii\_prod(data\_prod\_prev, ii\_prod.previous\_doc)

            else:

                if ii\_prod.next\_doc:

                    data\_word = {

                        "doc\_id\_first" : ii\_prod.next\_doc

                    }

                    m\_word = db\_handler.Get().ii\_word\_by\_doc\_id\_first(ii\_prod)

                    db\_handler.Update().ii\_words(data\_word, m\_word)

            if ii\_prod.next\_doc:

                if ii\_prod.previous\_doc:

                    data\_prod\_next = {

                        "previous\_doc" : ii\_prod.previous\_doc

                    }

                    db\_handler.Update().ii\_prod(data\_prod\_next, ii\_prod.next\_doc)

                else:

                    data\_prod\_next = {

                        "previous\_doc" : None

                    }

                    db\_handler.Update().ii\_prod(data\_prod\_next, ii\_prod.next\_doc)

            else:

                if ii\_prod.previous\_doc:

                    data\_word = {

                        "doc\_id\_last" : ii\_prod.previous\_doc,

                    }

                    m\_word = db\_handler.Get().ii\_word\_by\_doc\_id\_last(ii\_prod)

                    db\_handler.Update().ii\_words(data\_word, m\_word)

            data\_prod = {

                "previous\_doc" : None,

                "next\_doc" : None,

            }

            db\_handler.Update().ii\_prod(data\_prod, ii\_prod)

            # delete position

            m\_pos = ii\_prod.word\_position\_first

            while True:

                if not m\_pos.next:

                    m\_pos.delete()

                    break

                m\_pos.delete()

                m\_pos = m\_pos.next

            # delete doc\_id

            ii\_prod.delete()

    def create\_ii\_with\_doc\_id(self, words\_set, doc\_id):

        for word in words\_set:

            # delete doc\

            #  get word from InvertedIndexWordsTable

            m\_word = db\_handler.Get().word(word)

            for i, word\_pos in enumerate(words\_set[word]):

                if i == 0:

                    # create new pos

                    data\_pos = {

                        "position" : word\_pos

                    }

                    result\_pos, error\_pos = db\_handler.Insert().save\_inv\_indx\_word\_position(data\_pos)

                    # create doc

                    m\_prod\_id = db\_handler.Get().prod\_by\_prod\_id(doc\_id)

                    if m\_word:

                        data\_prod = {

                            "product\_id" : m\_prod\_id,

                            "word\_position\_first" : result\_pos["m\_ii\_word\_pos"],

                            "word\_position\_last" : result\_pos["m\_ii\_word\_pos"],

                            "previous\_doc" : m\_word.doc\_id\_last,

                        }

                    else:

                        data\_prod = {

                            "product\_id" : m\_prod\_id,

                            "word\_position\_first" : result\_pos["m\_ii\_word\_pos"],

                            "word\_position\_last" : result\_pos["m\_ii\_word\_pos"],

                        }

                    result\_prod, error\_prod = db\_handler.Insert().save\_ii\_prod(data\_prod)

                    # create word: InvertedIndexWordsTable

                    if m\_word and result\_prod["m\_ii\_prod"]:

                        if m\_word.doc\_id\_last:

                            data\_prod = {

                                "next\_doc" : result\_prod["m\_ii\_prod"],

                            }

                            result\_prod\_prev, error\_prod\_prev = db\_handler.Update().ii\_prod(data\_prod, m\_word.doc\_id\_last)

                            # update word last doc

                            data\_word = {

                                "doc\_id\_last" : result\_prod["m\_ii\_prod"],

                                # "docs\_count" : m\_word.docs\_count + 1,

                            }

                            db\_handler.Update().ii\_words(data\_word, m\_word)

                    else:

                        data\_word = {

                            "word" : word,

                            "doc\_id\_first" : result\_prod["m\_ii\_prod"],

                            "doc\_id\_last" : result\_prod["m\_ii\_prod"],

                        }

                        result\_word, error\_word = db\_handler.Insert().save\_ii\_word(data\_word)

                else:

                    # create pos

                    m\_prev\_pos = result\_pos["m\_ii\_word\_pos"]

                    data\_pos = {

                        "position" : word\_pos,

                        "previous" : m\_prev\_pos

                    }

                    result\_pos, error\_pos = db\_handler.Insert().save\_inv\_indx\_word\_position(data\_pos)

                    if i < len(words\_set[word]) :

                        # m\_new\_pos = result\_pos["m\_ii\_word\_pos"]

                        data\_pos = {

                            "next" : result\_pos["m\_ii\_word\_pos"]

                        }

                        db\_handler.Update().ii\_position(data\_pos, m\_prev\_pos)

                    # update doc

                    m\_ii\_prod = result\_prod["m\_ii\_prod"]

                    data\_prod = {

                                "word\_position\_last" : result\_pos["m\_ii\_word\_pos"],

                                "words\_count" : m\_ii\_prod.words\_count + 1,

                            }

                    result\_prod\_prev, error\_prod\_prev = db\_handler.Update().ii\_prod(data\_prod, m\_ii\_prod)

    def preprocess\_doc(self, doc):

        # extractcontraction

        contracted = contractions.fix(doc, leftovers=True, slang=True)

        # lemmatize

        doc\_lower = contracted.lower()

        sentence = sp(doc\_lower)

        doc\_token = [x.lemma\_ for x in sentence if not x.is\_punct]

        return doc\_token

    def save\_doc(self, data):

        result = {}

        error = []

        print("In invetrer index")

        print(data)

        doc\_id = data["doc\_id"]

        words\_token = self.preprocess\_doc(data["doc"])

        # create word: [pos... ] set

        words\_set = {}

        for word\_pos, word in enumerate(words\_token):

            if word in words\_set:

                pos = words\_set[word]

                pos.append(word\_pos)

                words\_set[word] = pos

            else:

                pos = [word\_pos]

                words\_set[word] = pos

        print(words\_set)

        """

        {

            we : [1, 5, 8],

            are : [10, 6, 2],

        }

        """

        self.delete\_doc\_id(doc\_id)

        self.create\_ii\_with\_doc\_id(words\_set, doc\_id)

        return result, error

    def sort\_dict(self, x):

        return {k: v for k, v in sorted(x.items(), key=lambda item: item[1], reverse=True)}

    def create\_prod\_html(self, doc\_score):

        html = ""

        for m\_prod in doc\_score:

            print(m\_prod)

            prod\_html, price = prod\_db.Get().get\_html\_one\_prod(m\_prod = m\_prod)

            html += prod\_html

        return html

    def search(self, data):

        query = data["query"]

        words\_token = self.preprocess\_doc(query)

        qf = dict()

        for word in words\_token:

            if word in qf:

                qf[word] = qf[word] + 1

            else:

                qf[word] = 1

        res = {}

        N = db\_handler.Get().products\_count()

        doc\_score = {}

        for i, word in enumerate(words\_token):

            m\_word = db\_handler.Get().word(word)

            prods = []

            if m\_word:

                # get docs of word

                m\_prod = m\_word.doc\_id\_first

                while True:

                    if m\_prod:

                        prods.append(m\_prod)

                    else:

                        break

                    m\_prod = m\_prod.next\_doc

                # no. of docs contain queries

                ni = len(prods)

                # calculate score of each prod

                print(prods)

                for m\_prd in prods:

                    fi = m\_prd.words\_count

                    qfi = qf[word]

                    score\_o = BestMatch25.BM25(N).Bm25(ni = ni, fi= fi, qfi = qfi)

                    score = round(score\_o, 2)

                    # print("word =", word, "ni =",  ni, "fi=", fi, "qfi =", qfi,"doc =", m\_prd.product\_id.id , "score = ", score)

                    doc\_id = m\_prd.product\_id

                    if doc\_id in doc\_score:

                        doc\_score[doc\_id] = doc\_score[doc\_id] + score

                    else:

                        doc\_score[doc\_id] = score

            # res[i] = {

            #         "word" : word,

            #         "prod" : prods

            #     }

        doc\_score = self.sort\_dict(doc\_score)

        # create result html

        html = self.create\_prod\_html(doc\_score)

        return html

## db\_handler.py

from SearchApp.forms import \*

from ProductsApp.models import ProductsTable

from django.forms.models import model\_to\_dict

from django.db.models import F

import pdb

class Get:

    def word(self, word):

        try:

            m\_word = InvertedIndexWordsTable.objects.get(word = word)

        except :

            m\_word = None

        return m\_word

    def prod\_by\_prod\_id(self, prod\_id):

        try:

            model\_prod\_id = ProductsTable.objects.get(id = prod\_id)

        except :

            model\_prod\_id = None

        return model\_prod\_id

    def all\_prod\_by\_prod\_id(self, prod\_id):

        m\_prod\_id = self.prod\_by\_prod\_id(prod\_id)

        return InvertedIndexProductTable.objects.filter(product\_id = m\_prod\_id)

    def ii\_word\_by\_doc\_id\_first(self, doc\_id\_first):

        try:

            m\_word = InvertedIndexWordsTable.objects.get(doc\_id\_first = doc\_id\_first)

        except :

            m\_word = None

        return m\_word

    def ii\_word\_by\_doc\_id\_last(self, doc\_id\_last):

        try:

            m\_word = InvertedIndexWordsTable.objects.get(doc\_id\_last = doc\_id\_last)

        except :

            m\_word = None

        return m\_word

    def products\_count(self):

        return ProductsTable.objects.all().count()

class Insert:

    def save\_inv\_indx\_word\_position(self, data):

        result = {}

        error = []

        form = InvertedIndexWordPositionTableForm(data)

        # form = InvertedIndexWordPositionTableForm(data)

        if form.is\_valid():

            m\_ii\_word\_pos = form.save(commit=True)

            result["is\_saved"] = True

            result["m\_ii\_word\_pos"] = m\_ii\_word\_pos

        if form.errors:

            result["is\_saved"] = False

            print(form.errors)

            result["m\_ii\_word\_pos"] = None

            error.append(form.errors)

        return result, error

    def save\_ii\_prod(self, data):

        result = {}

        error = []

        form = InvertedIndexProductTableForm(data)

        # form = InvertedIndexWordPositionTableForm(data)

        if form.is\_valid():

            m\_ii\_prod = form.save(commit=True)

            result["is\_saved"] = True

            result["m\_ii\_prod"] = m\_ii\_prod

        if form.errors:

            result["is\_saved"] = False

            print(form.errors)

            result["m\_ii\_prod"] = None

            error.append(form.errors)

        return result, error

    def save\_ii\_word(self, data):

        result = {}

        error = []

        form = InvertedIndexWordsTableForm(data)

        # form = InvertedIndexWordPositionTableForm(data)

        if form.is\_valid():

            m\_ii\_word = form.save(commit=True)

            result["is\_saved"] = True

            result["m\_ii\_word"] = m\_ii\_word

        if form.errors:

            result["is\_saved"] = False

            print(form.errors)

            result["m\_ii\_word"] = None

            error.append(form.errors)

        return result, error

class Update:

    def ii\_position(self, data, instance\_model):

        result = {}

        error = []

        data\_model = model\_to\_dict(instance\_model, fields=[field.name for field in instance\_model.\_meta.fields])

        data\_model.update(data)

        form = InvertedIndexWordPositionTableForm(data\_model, instance = instance\_model)

        # form = InvertedIndexWordPositionTableForm(data)

        if form.is\_valid():

            m\_ii\_word = form.save(commit=True)

            result["is\_saved"] = True

            result["m\_ii\_word"] = m\_ii\_word

        if form.errors:

            result["is\_saved"] = False

            print(form.errors)

            result["m\_ii\_word"] = None

            error.append(form.errors)

        return result, error

    def ii\_prod(self, data, instance\_model):

        result = {}

        error = []

        data\_model = model\_to\_dict(instance\_model, fields=[field.name for field in instance\_model.\_meta.fields])

        data\_model.update(data)

        form = InvertedIndexProductTableForm(data\_model, instance = instance\_model)

        if form.is\_valid():

            m\_ii\_prod = form.save(commit=True)

            result["is\_saved"] = True

            result["m\_ii\_prod"] = m\_ii\_prod

        if form.errors:

            result["is\_saved"] = False

            print(form.errors)

            result["m\_ii\_prod"] = None

            error.append(form.errors)

        return result, error

    def ii\_prod\_inc\_words\_count(self, data, instance\_model):

        result = {}

        error = []

        data\_model = model\_to\_dict(instance\_model, fields=[field.name for field in instance\_model.\_meta.fields])

        data\_model.update(data)

        data\_model["words\_count"] = data\_model["words\_count"] + 1

        form = InvertedIndexProductTableForm(data\_model, instance = instance\_model)

        if form.is\_valid():

            m\_ii\_word = form.save(commit=True)

            result["is\_saved"] = True

            result["m\_ii\_word"] = m\_ii\_word

        if form.errors:

            result["is\_saved"] = False

            print(form.errors)

            result["m\_ii\_word"] = None

            error.append(form.errors)

        return result, error

    def ii\_words(self, data, instance\_model):

        result = {}

        error = []

        data\_model = model\_to\_dict(instance\_model, fields=[field.name for field in instance\_model.\_meta.fields])

        data\_model.update(data)

        form = InvertedIndexWordsTableForm(data\_model, instance = instance\_model)

        if form.is\_valid():

            m\_ii\_word = form.save(commit=True)

            result["is\_saved"] = True

            result["m\_ii\_word"] = m\_ii\_word

        if form.errors:

            result["is\_saved"] = False

            print(form.errors)

            result["m\_ii\_word"] = None

            error.append(form.errors)

        return result, error

# class Delete:

#     def word\_position(self, id):

#         try:

#             InvertedIndexWordPositionTable.objects.get()

## Models.py

from django.db import models

from ProductsApp.models import ProductsTable

# InvertedIndex : word position

class InvertedIndexWordPositionTable(models.Model):

    id = models.AutoField( primary\_key = True)

    position = models.IntegerField(default=0)

    previous = models.ForeignKey("self", on\_delete=models.CASCADE, blank = True, null = True, related\_name="previous\_position")

    next = models.ForeignKey("self", on\_delete=models.CASCADE, blank = True, null = True, related\_name="next\_position")

    def \_\_str\_\_(self):

        return str(self.position) + " - " + str(self.id)

# InvertedIndex : product id

class InvertedIndexProductTable(models.Model):

    id = models.AutoField(primary\_key = True)

    product\_id = models.ForeignKey(ProductsTable, on\_delete=models.CASCADE)

    word\_position\_first = models.ForeignKey(InvertedIndexWordPositionTable, on\_delete=models.CASCADE, related\_name="first\_position")

    word\_position\_last = models.ForeignKey(InvertedIndexWordPositionTable, on\_delete=models.CASCADE, related\_name="last\_position")

    words\_count = models.IntegerField(default=1,blank=True, null=True)

    previous\_doc = models.ForeignKey("self", on\_delete=models.CASCADE, blank = True, null = True, related\_name="previous\_doc\_position")

    next\_doc = models.ForeignKey("self", on\_delete=models.CASCADE, blank = True, null = True, related\_name="next\_doc\_position")

    def \_\_str\_\_(self):

        return str(self.product\_id.id) + " - " + str(self.id)

# InvertedIndex : words

class InvertedIndexWordsTable(models.Model):

    word = models.CharField(max\_length = 150, primary\_key = True)

    doc\_id\_first = models.ForeignKey(InvertedIndexProductTable, on\_delete=models.CASCADE, related\_name="first\_position")

    doc\_id\_last = models.ForeignKey(InvertedIndexProductTable, on\_delete=models.CASCADE, related\_name="last\_position")

    docs\_count = models.IntegerField(default=1,blank=True, null=True)

    def \_\_str\_\_(self):

        return self.word

## forms.py

from django import forms

from SearchApp.models import \*

class InvertedIndexWordPositionTableForm(forms.ModelForm):

    class Meta:

        model = InvertedIndexWordPositionTable

        fields = '\_\_all\_\_'

class InvertedIndexProductTableForm(forms.ModelForm):

    class Meta:

        model = InvertedIndexProductTable

        fields = '\_\_all\_\_'

class InvertedIndexWordsTableForm(forms.ModelForm):

    class Meta:

        model = InvertedIndexWordsTable

        fields = '\_\_all\_\_'

## admin.py

from django.contrib import admin

from SearchApp.models import \*

admin.site.register(InvertedIndexWordPositionTable)

admin.site.register(InvertedIndexProductTable)

admin.site.register(InvertedIndexWordsTable)

# Compression

## Elias gamma encoder

https://www.geeksforgeeks.org/elias-gamma-encoding-in-python/

**from** math **import** log

log2 **=** **lambda** x: log(x, 2)

**def** Unary(x):

**return** (x**-**1)**\***'0'**+**'1'

**def** Binary(x, l **=** 1):

    s **=** '{0:0%db}' **%** l

**return** s.format(x)

**def** Elias\_Gamma(x):

**if**(x **==** 0):

**return** '0'

    n **=** 1 **+** int(log2(x))

    b **=** x **-** 2**\*\***(int(log2(x)))

    l **=** int(log2(x))

**return** Unary(n) **+** Binary(b, l)

print(Elias\_Gamma(10))

### decoder

https://www.geeksforgeeks.org/elias-gamma-decoding-in-python/

# import the math module

**import** math

# function

**def** Elias\_Gamma\_Decoding(x):

    x **=** list(x)

    K **=** 0

**while** True:

**if** **not** x[K] **==** '0':

**break**

        K **=** K **+** 1

    # Reading K more bits from '1'

    x **=** x[K:2**\***K**+**1]

    n **=** 0

    x.reverse()

    # Converting binary to integer

**for** i **in** range(len(x)):

**if** x[i] **==** '1':

            n **=** n**+**math.pow(2, i)

**return** int(n)

# value input

x **=** '0001001'

# call the function

print(Elias\_Gamma\_Decoding(x))

### Elias Delta encoding

https://www.geeksforgeeks.org/elias-delta-encoding-in-python/

**from** math **import** log

**from** math **import** floor

**def** Binary\_Representation\_Without\_MSB(x):

    binary **=** "{0:b}".format(int(x))

    binary\_without\_MSB **=** binary[1:]

**return** binary\_without\_MSB

**def** EliasGammaEncode(k):

**if** (k **==** 0):

**return** '0'

    N **=** 1 **+** floor(log(k, 2))

    Unary **=** (N**-**1)**\***'0'**+**'1'

**return** Unary **+** Binary\_Representation\_Without\_MSB(k)

**def** EliasDeltaEncode(x):

    Gamma **=** EliasGammaEncode(1 **+** floor(log(k, 2)))

    binary\_without\_MSB **=** Binary\_Representation\_Without\_MSB(k)

**return** Gamma**+**binary\_without\_MSB

k **=**  14

print(EliasDeltaEncode(k))

### decoding

<https://www.geeksforgeeks.org/elias-delta-decoding-in-python/?ref=rp>

**import** math

**def**  Elias\_Delta\_Decoding(x):

    x **=** list(x)

    L**=**0

**while** True:

**if** **not** x[L] **==** '0':

**break**

        L**=** L **+** 1

    # Reading L more bits and dropping ALL

    x**=**x[2**\***L**+**1:]

    # Prepending with 1 in MSB

    x.insert(0,'1')

    x.reverse()

    n**=**0

    # Converting binary to integer

**for** i **in** range(len(x)):

**if** x[i]**==**'1':

            n**=**n**+**math.pow(2,i)

**return** int(n)

x **=** '0111100'

print(Elias\_Delta\_Decoding(x))

# 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

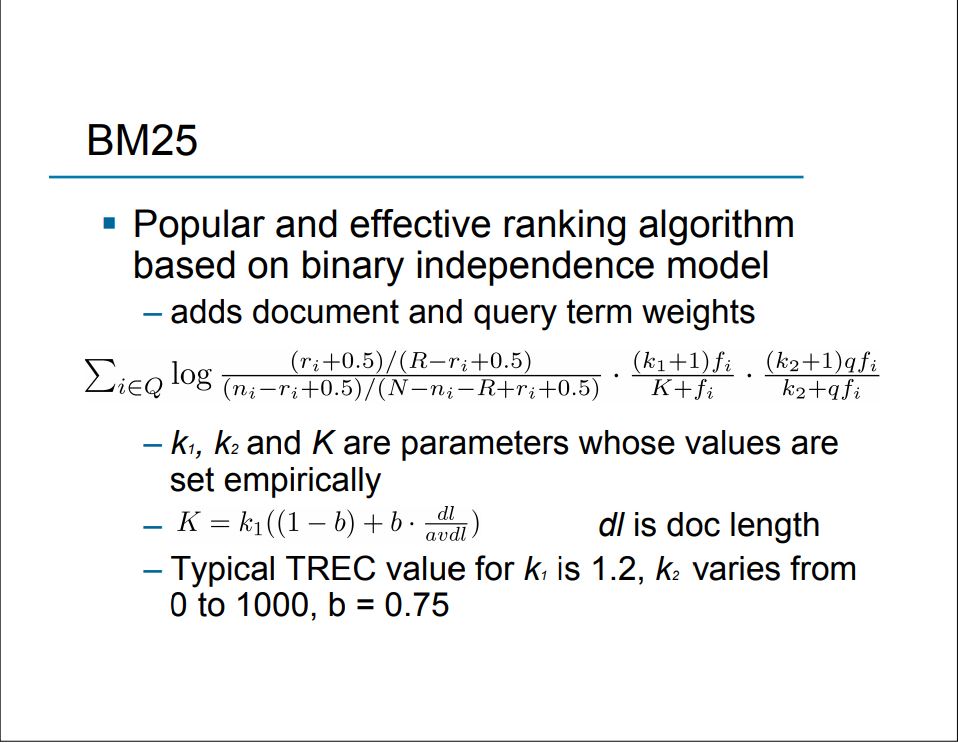
### Sort similarity

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

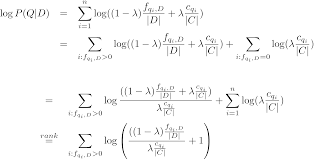
print(sims) # print sorted (document number, similarity score) 2-tuples

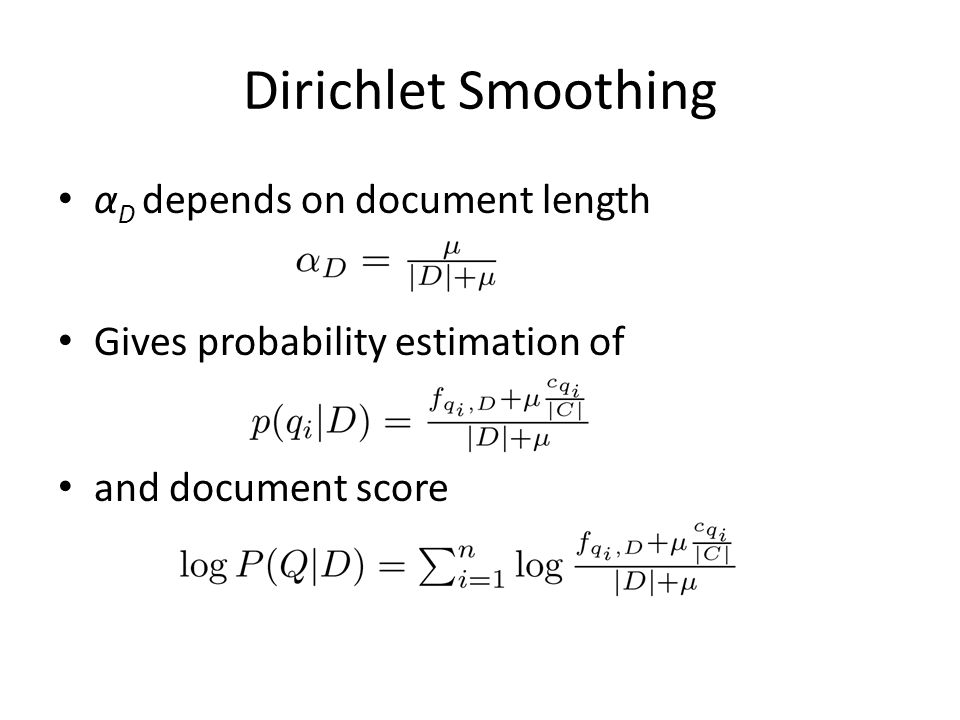
# Retrieval Models

## 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]

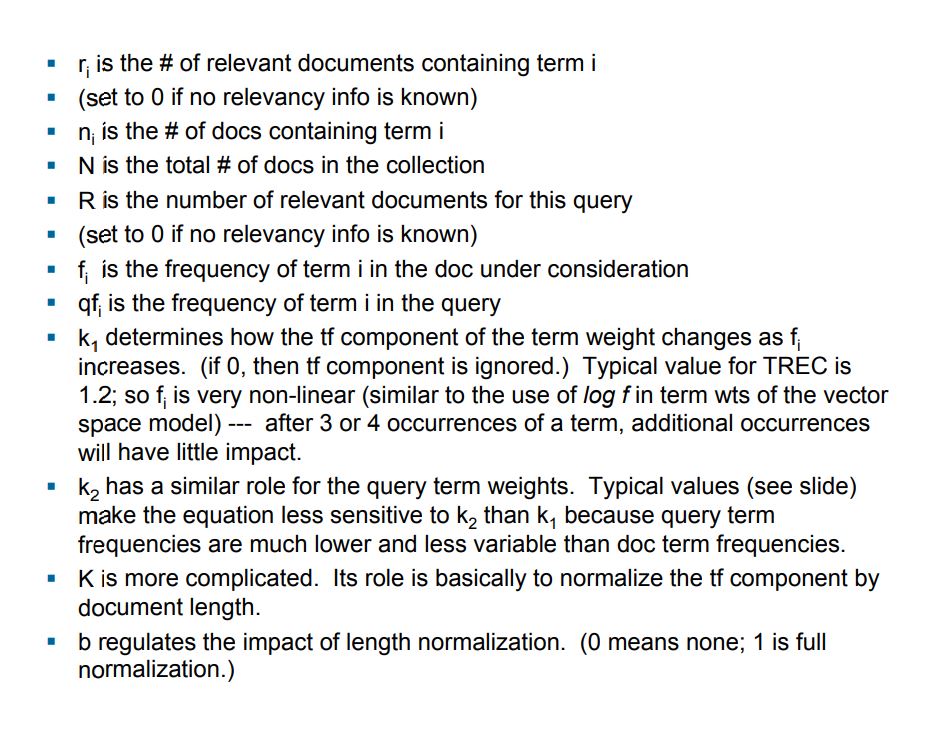
print(obj\_QL.doc\_score(fq, cq))

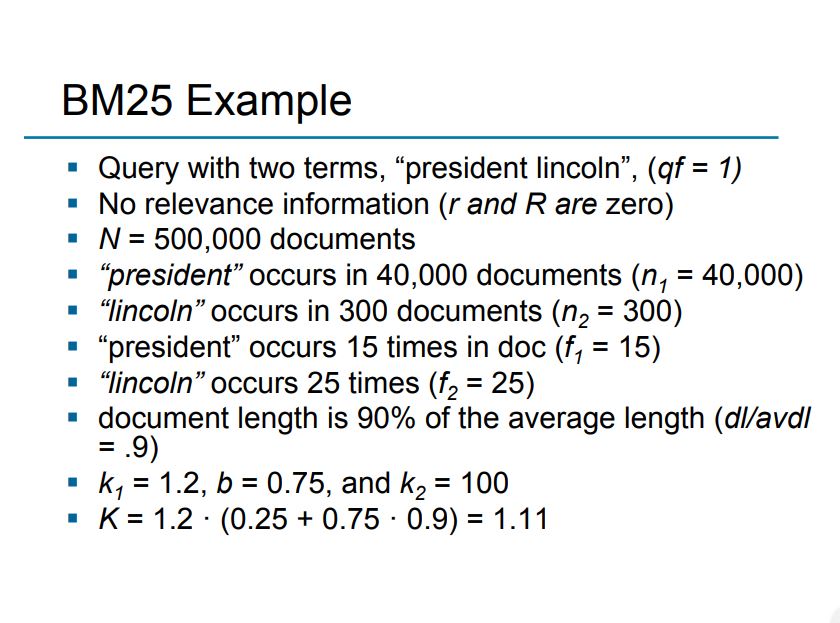
# output: -12.98

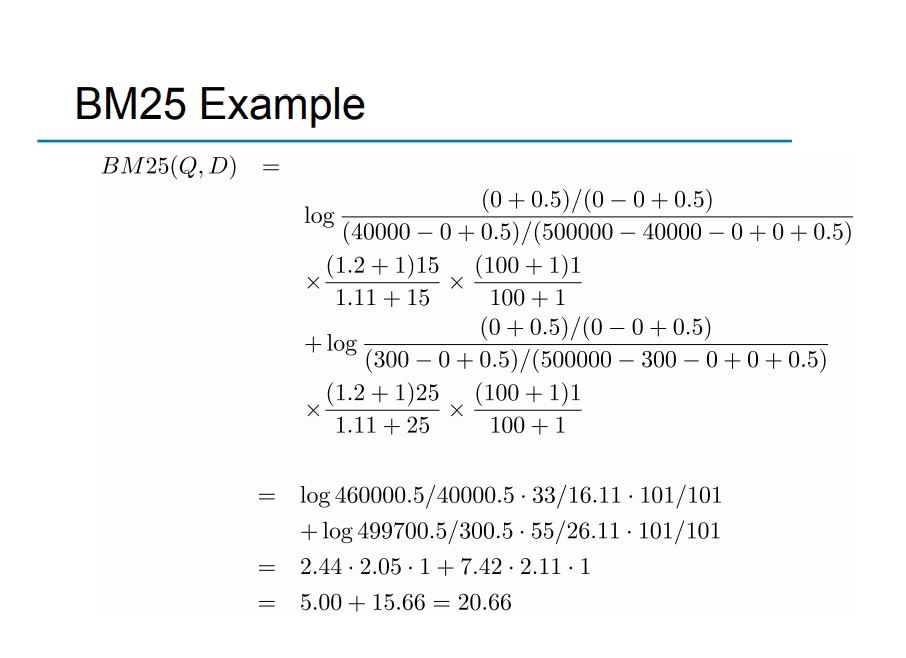
fq = [0, 25]

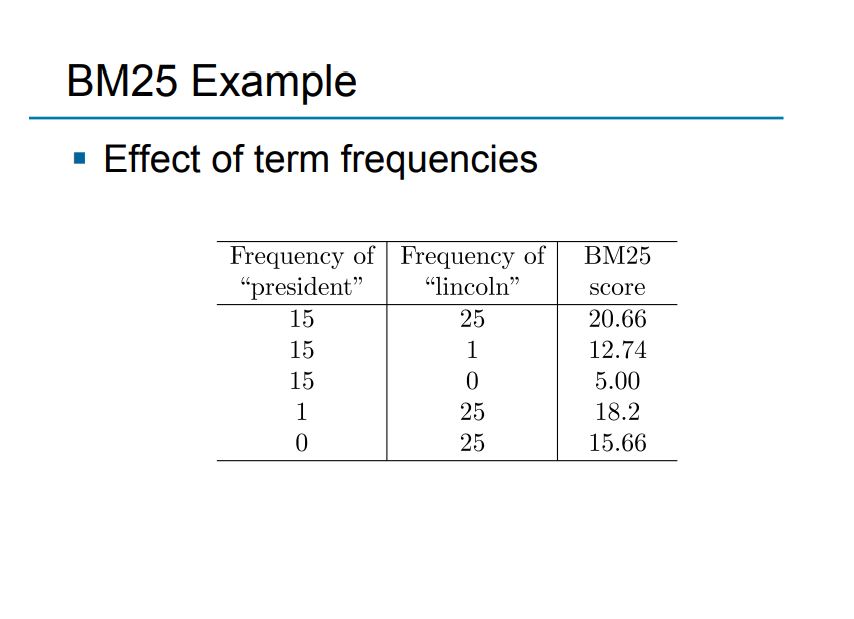
print(obj\_QL.doc\_score(fq, cq))

# 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/>

## BM25 Django

import numpy as np

import pdb

# R = 0

# N = 500000

# k1 = 1.2

# K = 1.11

# k2 = 100

class BM25:

    def \_\_init\_\_(self, N, R = 0, k1 = 1.2, K = 1.11, k2 = 100):

        """N is the total number of Documents"""

        self.R = R

        self.N = N

        self.k1 = k1

        self.K = K

        self.k2 = k2

    def PIdf(self, ri,ni):

        # pdb.set\_trace()

        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,  ni, fi, qfi, ri = 0):

        """

        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]

        """

        pidf = self.PIdf(ri, ni)

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

        return pidf \* norm\_f

## BM25 with relevance

### Elastic search - Bm25 with [medical ner](https://www.kaggle.com/code/ideanlabib/bm25-search-query-similarity-ranking)

#### ElasticSearch.py

*""""  
author: abhijith.m  
date created: 15-09-2022  
-------------------------  
reference: https://www.kaggle.com/code/ideanlabib/bm25-search-query-similarity-ranking/notebook  
"""*import numpy as np # linear algebra  
import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)  
from rank\_bm25 import BM25Okapi  
import nltk  
from nltk.corpus import stopwords  
# nltk.download("punkt")  
import re  
import spacy  
from tqdm import tqdm  
from scipy.spatial import distance  
nlp = spacy.load('en\_core\_sci\_sm')  
nlp.max\_length=2000000  
english\_stopwords = list(set(stopwords.words('english')))  
  
def strip\_characters(text):  
 t = re.sub('\(|\)|:|,|;|\.|’|”|“|\?|%|>|<', '', text)  
 t = re.sub('/', ' ', t)  
 t = t.replace("'",'')  
 return t  
  
def clean(text):  
 # t = text.lower()  
 t = strip\_characters(text)  
 return t  
  
def tokenize(text):  
 words = nltk.word\_tokenize(text)  
 return list(set([word for word in words  
 if len(word) > 1  
 and not word in english\_stopwords  
 and not (word.isnumeric() and len(word) is not 4)  
 and (not word.isnumeric() or word.isalpha())] )  
 )  
  
def preprocess(text):  
 t = clean(text)  
 tokens = tokenize(t)  
 return tokens  
  
class SearchResults:  
  
 def \_\_init\_\_(self,  
 data: pd.DataFrame,  
 columns = None):  
 self.results = data  
 if columns:  
 self.results = self.results[columns]  
  
 def \_\_getitem\_\_(self, item):  
 return Paper(self.results.loc[item])  
  
 def \_\_len\_\_(self):  
 return len(self.results)  
  
 def \_repr\_html\_(self):  
 return self.results.\_repr\_html\_()  
  
SEARCH\_DISPLAY\_COLUMNS = ['\_id', 'Query']  
  
class WordTokenIndex:  
  
 def \_\_init\_\_(self,  
 corpus: pd.DataFrame,  
 columns=SEARCH\_DISPLAY\_COLUMNS):  
 self.corpus = corpus  
 raw\_search\_str = self.corpus.Query.fillna('')  
 self.index = raw\_search\_str.apply(preprocess).to\_frame()  
 self.index.columns = ['terms']  
 self.index.index = self.corpus.index  
 self.columns = columns  
  
 def search(self, search\_string):  
 search\_terms = preprocess(search\_string)  
 result\_index = self.index.terms.apply(lambda terms: any(i in terms for i in search\_terms))  
 results = self.corpus[result\_index].copy().reset\_index().rename(columns={'index':'paper'})  
 return SearchResults(results, self.columns + ['paper'])  
  
class RankBM25Index(WordTokenIndex):  
  
 def \_\_init\_\_(self, corpus: pd.DataFrame, columns=SEARCH\_DISPLAY\_COLUMNS):  
 super().\_\_init\_\_(corpus, columns)  
 self.bm25 = BM25Okapi(self.index.terms.tolist())  
  
 def search(self, search\_string, n=4, weightage=1):  
 search\_terms = preprocess(search\_string)  
 doc\_scores = self.bm25.get\_scores(search\_terms)  
 ind = np.argsort(doc\_scores)[::-1][:n]  
 results = self.corpus.iloc[ind][self.columns]  
 results['Score'] = doc\_scores[ind] \* weightage  
 results['orig\_ind'] = ind  
 results = results[results.Score > 0]  
 return SearchResults(results.reset\_index(), self.columns + ['Score', 'orig\_ind'])  
  
  
class RankBM25Score:  
 def \_\_init\_\_(self, bm25\_index, \*args, \*\*kwargs):  
 self.bm25\_index = bm25\_index  
  
 def score(self, keywords):  
 results = None  
 added = []  
 for i, word in enumerate(keywords):  
 weightage = 1 + ((len(keywords) - i)/10)  
 word\_result = self.bm25\_index.search(word, n=100, weightage=weightage).results  
 if results is None:  
 results = word\_result  
 added += [r.orig\_ind for i, r in word\_result.iterrows()]  
 continue  
 for i, r in word\_result.iterrows():  
 if r.orig\_ind not in added:  
 results = results.append(r)  
 added.append(r.orig\_ind)  
 df = results.sort\_values(by='Score', ascending=False)  
 df = df.dropna()  
 df.reset\_index(drop=True, inplace=True)  
 return df  
  
class ElasticSearch:  
  
 def vectorizing(self, df):  
 vector\_list = []  
 for i in tqdm(df.index):  
 doc = nlp(df.iloc[i].Query)  
 sents = [sent for sent in doc.sents]  
 vecs = [sent.vector for sent in sents]  
 for j in range(len(sents)):  
 vector\_list.append(  
 {"\_id": df.iloc[i].\_id,  
 "score": df.iloc[i]['Score'],  
 "sentence": j,  
 "vector": vecs[j],  
 "start\_span": sents[j].start\_char,  
 "end\_span": sents[j].end\_char})  
 vector\_df = pd.DataFrame(data=vector\_list)  
 return vector\_df  
  
 def query\_vec\_creation(self, queries, i):  
 query\_vector\_list = []  
 doc = nlp(queries)  
 vec = doc.vector  
 query\_vector\_list.append({"\_id": f"query\_{i}", "vector": vec})  
 query\_vector\_df = pd.DataFrame(data=query\_vector\_list)  
 return query\_vector\_df  
  
 def w2v\_searchable\_df\_creation(self, RATIO, vector\_df, query\_vector\_df, i):  
 distances = distance.cdist([value for value in query\_vector\_df["vector"]], [value for value in vector\_df["vector"].values], "cosine")  
 w2v\_searchable\_df = vector\_df.drop(columns=["vector"])  
 # Create a column with cosine distances for each query vs the sentence  
 w2v\_searchable\_df[f"query\_{i}\_distance"] = RATIO \* (1 - distances[0]) + (1-RATIO) \* w2v\_searchable\_df['score']  
 return w2v\_searchable\_df  
  
 def result\_df\_creation(self, df,w2v\_searchable\_df, i):  
 columnName = f"query\_{i}\_distance"  
 context = w2v\_searchable\_df.sort\_values(by=columnName, ascending=False)[["\_id","start\_span","end\_span"]][:20]  
 ix = context["\_id"].to\_list()  
 spans1 = context["start\_span"].to\_list()  
 spans2 = context["end\_span"].to\_list()  
 result = []  
 for j in range(len(context.index)):  
 score = df[df["\_id"] == ix[j]].iloc[0]['Score']  
 id = df[df["\_id"] == ix[j]].iloc[0]['\_id']  
 row = {  
 "id": id,  
 "Insight": str(df[df["\_id"] == ix[j]].iloc[0]["Query"])[spans1[j]:spans2[j]],  
 # "rank": j+1,  
 "Relevance Score": score  
 }  
 result.append(row)  
 df\_out = pd.DataFrame.from\_records(result)  
 return df\_out

#### Main\_streamlit.py

import streamlit as st  
from ClusteringBase import Clustering  
from PreprocessTfidf import preprocess\_insights  
import pandas as pd  
from st\_aggrid import GridOptionsBuilder, AgGrid  
from ElasticSearch import RankBM25Index, RankBM25Score, ElasticSearch  
  
  
def k\_mean\_df\_creation(num\_cluster, df\_prepro, df\_inp):  
 clstr\_obj = Clustering()  
 df\_prepro = df\_prepro.dropna()  
 df\_prepro = df\_prepro["Query"]  
 documents = df\_prepro.values.astype("U")  
 # Vectorizing  
 terms, X = clstr\_obj.vectorizing(documents)  
 cluster\_labels, clusterer = clstr\_obj.clustering(num\_cluster, X)  
 # silhouette\_score\_cal  
 sample\_silhouette\_values, avg\_each\_label = clstr\_obj.silhouette\_score\_cal(cluster\_labels, X)  
 # keyword\_cal  
 clusters, exclude\_cluster = clstr\_obj.keyword\_cal(clusterer, num\_cluster, terms)  
 # training\_bar.progress(25)  
 # insightNKeywords  
 clusr\_keywords = clstr\_obj.insightNKeywords(cluster\_labels, clusters, len(documents))  
 # training\_bar.progress(50)  
  
 df\_inp['cluster'] = cluster\_labels  
 df\_inp['distance score'] = sample\_silhouette\_values  
 df\_inp['cluster rank(0-low, %d-high)'%(num\_cluster-1)] = avg\_each\_label  
 return df\_inp, clusr\_keywords  
  
def bm25\_df\_creation(keywords, df\_prepro):  
 bm25\_index = RankBM25Index(df\_prepro)  
 df = RankBM25Score(bm25\_index).score(keywords)  
 return df  
  
def elastic\_search(keywords, df\_prepro, tab, semantic\_weightage):  
 df = bm25\_df\_creation(keywords, df\_prepro)  
 obj\_elastic\_search = ElasticSearch()  
 vector\_df = obj\_elastic\_search.vectorizing(df)  
 query\_vector\_df = obj\_elastic\_search.query\_vec\_creation(" ".join(keywords), tab)  
 w2v\_searchable\_df = obj\_elastic\_search.w2v\_searchable\_df\_creation(semantic\_weightage, vector\_df, query\_vector\_df, tab)  
 out\_df = obj\_elastic\_search.result\_df\_creation(df,w2v\_searchable\_df,tab)  
 return out\_df  
  
# section 1  
# ########################################################################  
st.set\_page\_config(  
 page\_title="Insight Clustering",  
 page\_icon="🎯",  
 layout="wide",  
 )  
sec1 = st.columns([1,1,1,1])  
sec1[0].title("Clustering")  
model\_sel = sec1[1].radio("Choose model", ("K-mean", "K-mean + BM25", "K-mean + elasticsearch"))  
sec1[2].header("Training")  
uploaded\_file = sec1[3].file\_uploader("Dataset")  
if uploaded\_file is not None:  
 df\_inp = pd.read\_csv(uploaded\_file)  
 df\_prepro = preprocess\_insights(df=df\_inp)  
 num\_clstr = sec1[3].number\_input("Number of clusters:", min\_value=2)  
 btn\_train = sec1[3].button("Start training")  
  
# section 2  
# ########################################################################  
  
 df\_kmean, keywords\_kmean = k\_mean\_df\_creation(int(num\_clstr), df\_prepro, df\_inp)  
 clstr\_sel\_tab = st.tabs(["Cluster "+str(x) for x in range(int(num\_clstr))])  
 if model\_sel == "K-mean + elasticsearch":  
 semantic\_weightage = st.slider("Weightage of semantic search", 0.1, 4.0, 0.9, 0.1, key="sematic\_wt")  
 for i, tab in enumerate(clstr\_sel\_tab):  
 with tab:  
 tab\_cols = tab.columns([2,1])  
 df\_keyword = pd.DataFrame({"Keywords": pd.Series(keywords\_kmean[i])})  
 if model\_sel == "K-mean":  
 df\_show\_kmn = df\_kmean[df\_kmean.cluster == i]  
 df\_show\_kmn = df\_show\_kmn.sort\_values(by=["distance score"], ascending=False)  
 tab\_cols[0].dataframe(df\_show\_kmn, height=200)  
  
 # tab\_cols[1].dataframe(pd.DataFrame({"Keywords": pd.Series(keywords\_kmean[i])}), height=200)  
 with tab\_cols[1]:  
 options\_builder = GridOptionsBuilder.from\_dataframe(df\_keyword)  
 options\_builder.configure\_column('Keywords', editable=True)  
 # options\_builder.configure\_pagination(paginationAutoPageSize=True)  
 options\_builder.configure\_selection("single")  
 # options\_builder.configure\_grid\_options(rowHeight=50)  
 grid\_options = options\_builder.build()  
  
 grid\_return = AgGrid(df\_keyword, grid\_options, theme='streamlit', data\_return\_mode='as\_input', update\_mode='MODEL\_CHANGED', height=250, key="agd%d"%i)  
 selected\_rows = grid\_return["selected\_rows"]  
 elif model\_sel == "K-mean + BM25":  
 df\_show\_bm25 = bm25\_df\_creation(keywords\_kmean[i], df\_prepro)  
 tab\_cols[0].dataframe(df\_show\_bm25, height=200)  
 tab\_cols[1].dataframe(pd.DataFrame({"Keywords": pd.Series(keywords\_kmean[i])}), height=200)  
 elif model\_sel == "K-mean + elasticsearch":  
 df\_show\_elastic = elastic\_search(keywords\_kmean[i], df\_prepro, i, int(semantic\_weightage))  
 tab\_cols[0].dataframe(df\_show\_elastic, height=200)  
 tab\_cols[1].dataframe(pd.DataFrame({"Keywords": pd.Series(keywords\_kmean[i])}), height=200)

### bm25 [with bert](https://medium.com/@papai143/information-retrieval-with-document-re-ranking-with-bert-and-bm25-7c29d738df73)

# 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

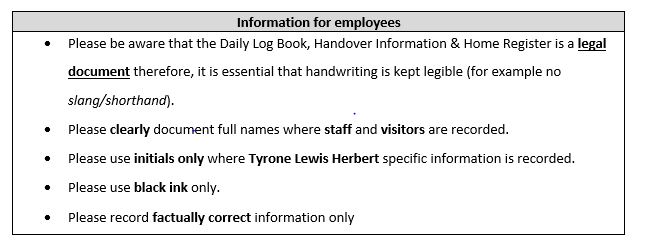
1. github code: <https://github.com/carpedm20/MemN2N-tensorflow>

* 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/>

# Word suggestor

from difflib import get\_close\_matches

def search\_command():

    try:

        book\_list = ["Fundamentals of Physics Class XII",

                    "A Textbook of Engineering Physics",

                    "Introduction to Sustainable Engineering",

                    "ഓർമശക്തി ഇരട്ടിയാക്കാം",

                    "Theory of Computations Part 1"]

        word = input("Enter a word: ")

        len\_close\_match = 0

        dyn\_cut\_off = 0.9

        while len\_close\_match == 0 and dyn\_cut\_off != 0.0:

            close\_match = get\_close\_matches(word, book\_list, cutoff= dyn\_cut\_off)

            len\_close\_match = len(close\_match)

            dyn\_cut\_off-= .1

            # print("%.1f"%cut\_off) #display only single decimal point

            print(dyn\_cut\_off)

        print("Did you mean:\n")

        if len\_close\_match == 1.0:

            print(close\_match[0])

        else:

            print(close\_match[:2])

    except IndexError:

        print("Error")

search\_command()

# Reference

## Sentiment analyisis

### Word2Vec + LSTM

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

### best libraries

<https://github.com/cjhutto/vaderSentiment>

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

# 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>

## With pretrained model

<https://neptune.ai/blog/document-classification-small-datasets>

fasttext pretrained [model](https://fasttext.cc/docs/en/english-vectors.html)

## Document classification youtube

### Invoice to vec – not working

[Youtube](https://www.youtube.com/watch?v=18Pxvs50G-0)

# Auto-complete

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

# Doc2Vec Training

from gensim.models import Doc2Vec

from gensim.models.doc2vec import TaggedDocument

from sklearn.neighbors import KNeighborsClassifier

from sklearn import svm

from sklearn.ensemble import RandomForestClassifier

from InvoiceApp.forms import \*

import os

from sklearn import preprocessing

import pickle

print("Please note after training you should need to restart server")

path\_model = os.path.join("model")

if not os.path.exists(path\_model):

    os.mkdir(path\_model)

window = 4

path\_word2vec\_model = os.path.join(path\_model,'model\_docs\_%s.doc2vec'%(window))

path\_TaskCode = os.path.join(path\_model,'TaskCode.pickle')

path\_TaskActivityCode = os.path.join(path\_model, "TaskActivityCode.pickle")

path\_TKcode = os.path.join(path\_model, "TKcode.pickle")

path\_TKLevel = os.path.join(path\_model, "TKLevel.pickle")

path\_PayerReductionReason = os.path.join(path\_model, "PayerReductionReason.pickle")

path\_knn\_model = os.path.join(path\_model, "knn\_model.pickle")

path\_svm\_model = os.path.join(path\_model, "svm\_model.pickle")

path\_randomforest\_model = os.path.join(path\_model, "randomforest\_model.pickle")

path\_description = os.path.join("data/ChargeDescriptionFullData/")

class TaggedDocumentIterator(object):

    def \_\_init\_\_(self, doc\_list, labels\_list):

        self.labels\_list = labels\_list

        self.doc\_list = doc\_list

    def \_\_iter\_\_(self):

        for idx, doc in enumerate(self.doc\_list):

            yield TaggedDocument(words=doc.split(), tags=[self.labels\_list[idx]])

class Preprocess:

    def clean\_features(self):

        print("Start cleaning data ...")

        docLabels = []

        Units = []

        Rate = []

        TotalAmount = []

        TaskCode = []

        TaskActivityCode = []

        TKcode = []

        TKLevel = []

        PayerReductionReason = []

        inst\_obj = DatasetDB.objects.all().order\_by('id')

        for insight\_obj in inst\_obj:

            TaskCode.append(insight\_obj.TaskCode)

            TaskActivityCode.append(insight\_obj.TaskActivityCode)

            TKcode.append(insight\_obj.TKcode)

            TKLevel.append(insight\_obj.TKLevel)

            PayerReductionReason.append(insight\_obj.PayerReductionReason)

        # label encode string values

        le = preprocessing.LabelEncoder()

        le.fit(TaskCode)

        f = open(path\_TaskCode, 'wb')

        pickle.dump(le, f)

        f.close()

        le1 = preprocessing.LabelEncoder()

        le1.fit(TaskActivityCode)

        f1 = open(path\_TaskActivityCode, 'wb')

        pickle.dump(le1, f1)

        f1.close()

        le2 = preprocessing.LabelEncoder()

        le2.fit(TKcode)

        f2 = open(path\_TKcode, 'wb')

        pickle.dump(le2, f2)

        f2.close()

        le3 = preprocessing.LabelEncoder()

        le3.fit(TKLevel)

        f3 = open(path\_TKLevel, 'wb')

        pickle.dump(le3, f3)

        f3.close()

        le4 = preprocessing.LabelEncoder()

        le4.fit(PayerReductionReason)

        f4 = open(path\_PayerReductionReason, 'wb')

        pickle.dump(le4, f4)

        f4.close()

        print("Cleaning compled ...")

class TrainingInsight:

    def check\_float(self, num):

        try :

            float(num)

            res = True

        except :

            res = False

        return res

    def train\_word2vec(self):

        # start training

        print("start training word2vec...")

        insights = []

        docLabels = []

        inst\_obj = DatasetDB.objects.all().order\_by('id')

        for insight\_obj in inst\_obj:

            docLabels.append(insight\_obj.id)

            insights.append(insight\_obj.ChargeDescription)

        sentences = TaggedDocumentIterator(insights, docLabels)

        model = Doc2Vec(vector\_size=100, window=window, min\_count=5, workers=11,alpha=0.025, epochs=20)

        model.build\_vocab(sentences)

        model.train(sentences,total\_examples=model.corpus\_count, epochs=model.iter)

        model.save(path\_word2vec\_model)

        print ("%s trained successfully"%(window))

        return True

    def load\_training\_data(self):

        ChargeDescription = []

        Units = []

        Rate = []

        TotalAmount = []

        TaskCode = []

        TaskActivityCode = []

        TKcode = []

        TKLevel = []

        PayerReductionReason = []

        # load word2vec description

        file = open(path\_TaskCode, 'rb')

        le\_TaskCode = pickle.load(file)

        file.close()

        file1 = open(path\_TaskActivityCode, 'rb')

        le\_TaskActivityCode = pickle.load(file1)

        file1.close()

        file2 = open(path\_TKcode, 'rb')

        le\_TKcode = pickle.load(file2)

        file2.close()

        file3 = open(path\_TKLevel, 'rb')

        le\_TKLevel = pickle.load(file3)

        file3.close()

        file4 = open(path\_PayerReductionReason, 'rb')

        le\_PayerReductionReason = pickle.load(file4)

        file4.close()

        model\_word2vec = Doc2Vec.load(path\_word2vec\_model)

        inst\_obj = DatasetDB.objects.all().order\_by('id')

        X = []

        for insight\_obj in inst\_obj:

            tmp = []

            vector = model\_word2vec.infer\_vector(insight\_obj.ChargeDescription.split())

            # ChargeDescription.append(vector)

            for x in vector:

                tmp.append(x)

            if self.check\_float(str(insight\_obj.Units)):

                # Units.append(insight\_obj.Units)

                tmp.append(float(insight\_obj.Units))

            else:

                # Units.append(0)

                tmp.append(0)

            if self.check\_float(str(insight\_obj.Rate)):

                # Rate.append(insight\_obj.Rate)

                tmp.append(float(insight\_obj.Rate))

            else:

                # Rate.append(0)

                tmp.append(0)

            if self.check\_float(str(insight\_obj.TotalAmount)):

                # TotalAmount.append(insight\_obj.TotalAmount)

                tmp.append(float(insight\_obj.TotalAmount))

            else:

                # TotalAmount.append(0)

                tmp.append(0)

            # TaskCode.append(le\_TaskCode.transform([insight\_obj.TaskCode] ))

            tmp.append(le\_TaskCode.transform([insight\_obj.TaskCode] )[0])

            # TaskActivityCode.append(le\_TaskActivityCode.transform([insight\_obj.TaskActivityCode] ))

            tmp.append(le\_TaskActivityCode.transform([insight\_obj.TaskActivityCode])[0])

            # TKcode.append(le\_TKcode.transform([insight\_obj.TKcode] ))

            tmp.append(le\_TKcode.transform([insight\_obj.TKcode])[0])

            # TKLevel.append(le\_TKLevel.transform([insight\_obj.TKLevel] ))

            tmp.append(le\_TKLevel.transform([insight\_obj.TKLevel] )[0])

            PayerReductionReason.append(le\_PayerReductionReason.transform([insight\_obj.PayerReductionReason] )[0])

            # tmp.append(le\_PayerReductionReason.transform([insight\_obj.PayerReductionReason]))

            X.append(tmp)

            # print(tmp)

            # break

        # X.append(ChargeDescription)

        # X.append(Units)

        # X.append(Rate)

        # X.append(TotalAmount)

        # X.append(TaskCode)

        # X.append(TaskActivityCode)

        # X.append(TKcode)

        # X.append(TKLevel)

        return X, PayerReductionReason

    def train\_knn(self, X, y):

        neigh = KNeighborsClassifier(n\_neighbors=3)

        neigh.fit(X, y)

        f = open(path\_knn\_model, 'wb')

        pickle.dump(neigh, f)

        f.close()

        print("Training KNN completed...")

    def train\_svm(self, X, y):

        clf = svm.SVC()

        clf.fit(X, y)

        f = open(path\_svm\_model, 'wb')

        pickle.dump(clf, f)

        f.close()

        print("Training SVM completed...")

    def train\_randomforest(self, X, y):

        clf = RandomForestClassifier(max\_depth=6, random\_state=0)

        clf.fit(X, y)

        f = open(path\_randomforest\_model, 'wb')

        pickle.dump(clf, f)

        f.close()

        print("Training Randomforest completed...")

    def train(self):

        # self.train\_word2vec()

        # cleaning features

        # Preprocess().clean\_features()

        X, y = self.load\_training\_data()

        print(len(X), len(y))

        print(X[0], y[0])

        print("start training")

        # self.train\_knn(X, y)

        # self.train\_svm(X, y)

        self.train\_randomforest(X, y)

clf\_randomforest = None

class Prediction:

    def load\_knn\_model(self):

        file = open(path\_knn\_model, 'rb')

        data = pickle.load(file)

        file.close()

        return data

    def load\_svm\_model(self):

        return 0

    def load\_randomforest\_model(self):

        file = open(path\_randomforest\_model, 'rb')

        data = pickle.load(file)

        file.close()

        return data

    def predict(self, data):

        result = {}

        tmp = []

        vector = model\_word2vec.infer\_vector(data["ChargeDescription"].split())

        for x in vector:

            tmp.append(x)

        if TrainingInsight().check\_float(str(data["Units"])):

            # Units.append(insight\_obj.Units)

            tmp.append(float(data["Units"]))

        else:

            # Units.append(0)

            tmp.append(0)

        if TrainingInsight().check\_float(str(data["Rate"])):

            # Rate.append(insight\_obj.Rate)

            tmp.append(float(data["Rate"]))

        else:

            # Rate.append(0)

            tmp.append(0)

        if TrainingInsight().check\_float(str(data["TotalAmount"])):

            # TotalAmount.append(insight\_obj.TotalAmount)

            tmp.append(float(data["TotalAmount"]))

        else:

            # TotalAmount.append(0)

            tmp.append(0)

        # TaskCode.append(le\_TaskCode.transform([insight\_obj.TaskCode] ))

        tmp.append(le\_TaskCode.transform([data["TaskCode"]] )[0])

        # TaskActivityCode.append(le\_TaskActivityCode.transform([insight\_obj.TaskActivityCode] ))

        tmp.append(le\_TaskActivityCode.transform([data["TaskActivityCode"]])[0])

        # TKcode.append(le\_TKcode.transform([insight\_obj.TKcode] ))

        tmp.append(le\_TKcode.transform([data["TKcode"]])[0])

        # TKLevel.append(le\_TKLevel.transform([insight\_obj.TKLevel] ))

        tmp.append(le\_TKLevel.transform([data["TKLevel"]] )[0])

        if clf\_randomforest:

            predicted\_o = clf\_randomforest.predict([tmp])

            predicted = clf\_randomforest.predict\_proba([tmp])

            score = predicted[0][predicted\_o[0]]

            out = le\_PayerReductionReason.inverse\_transform(predicted\_o)

            result["predicted"] = out[0]

            result["probability"] = score

        print(result)

        return result

try:

    file = open(path\_TaskCode, 'rb')

    le\_TaskCode = pickle.load(file)

    file.close()

    file1 = open(path\_TaskActivityCode, 'rb')

    le\_TaskActivityCode = pickle.load(file1)

    file1.close()

    file2 = open(path\_TKcode, 'rb')

    le\_TKcode = pickle.load(file2)

    file2.close()

    file3 = open(path\_TKLevel, 'rb')

    le\_TKLevel = pickle.load(file3)

    file3.close()

    file4 = open(path\_PayerReductionReason, 'rb')

    le\_PayerReductionReason = pickle.load(file4)

    file4.close()

    model\_word2vec = Doc2Vec.load(path\_word2vec\_model)

    clf\_randomforest = Prediction().load\_randomforest\_model()

except :

    print("Model is not trained...")

# Named entity recogniton

<https://spacy.io/usage/training>

* git clone <https://github.com/UniversalDependencies/UD_Spanish-AnCora.git>
* mkdir ancora-json
* python -m spacy convert UD\_Spanish-AnCora/es\_ancora-ud-train.conllu ancora-json
* python -m spacy convert UD\_Spanish-AnCora/es\_ancora-ud-dev.conllu ancora-json
* mkdir models
* python -m spacy train es models ancora-json/es\_ancora-ud-train.json ancora-json/es\_ancora-ud-dev.json

## Named entity labeling

<https://prodi.gy/>

# Scispacy

<https://allenai.github.io/scispacy/>

if model not installing download models from -> [here](https://pythonrepo.com/repo/allenai-scispacy-python-natural-language-processing)

## linguistic feature

<https://spacy.io/usage/linguistic-features>

## Custom training

<https://www.machinelearningplus.com/nlp/training-custom-ner-model-in-spacy/>

# nlp with python book

* Re.findall()

### index

Nltk.Index()

# One-hot encode numpy

article\_title = tweet["Article Title"].values

abstract\_text = tweet["Abstract Text"].values

tag = tweet["Tag"].values

le = preprocessing.LabelEncoder()

tag\_le = le.fit\_transform(tag)

classes\_le = le.classes\_

# one-hot encoding

a = np.array([le.transform([x])[0] for x in classes\_le])

b = np.zeros((a.size, a.max()+1))

b[np.arange(a.size),a] = 1

ohe\_tag = []

for t in tag\_le:

    ohe\_tag.append(b[t])

# Relevance-feedback-and-Query-expansion

<https://github.com/siddumsp1997/Relevance-feedback-and-Query-expansion>

good code: <https://github.com/mohit155/SearchEngine>

# Bert

BertViz

# Reading Comprehension

## cdqa

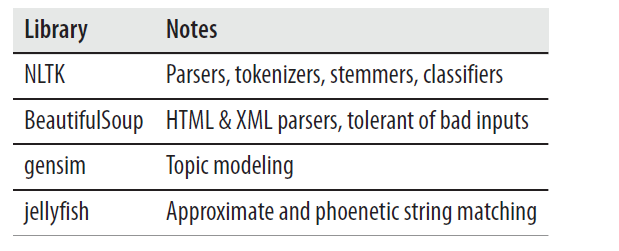
Clone and install requirements. Example folder has ipynb files

https://github.com/cdqa-suite/cdQA.git

# PDF processing

* pip install PyMuPDF==1.16.14
* [documentation](https://pymupdf.readthedocs.io/en/latest/tutorial.html#importing-the-bindings)

# Text Processing



## Chickenfoot

which is a Firefox plug-in that allows you to programmatically interact with a web page through the browser

## nltk-trainer

is an open source library

of scripts I created for training and analyzing NLTK models

## MEGAM

algorithm for training a MaxentClassifi

er using each review paragraph as a single instance, looking at both single words (unigrams)

and pairs of words (bigrams). The MaxentClassifier (or Logistic Regression),

uses an iterative algorithm to determine weights for every feature.

# Topic modeling

## Corex

<https://github.com/gregversteeg/corex_topic>

<https://medium.com/version-1/exploring-topic-modelling-using-semi-supervised-learning-correlation-explanation-b81d2603c9a2>

# Text Summarization