```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

1. Preprocessing of raw data

```
import nltk
import numpy as np
import pandas as pd
```

```
nltk.download("punkt")
nltk.download("stopwords")
nltk.download("wordnet")
nltk.download('omw-1.4')
from nltk.tokenize import word_tokenize
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
```

```
stopwords = nltk.corpus.stopwords.words('english')
stemmer = nltk.stem.SnowballStemmer('english')
lemmatizer = nltk.stem.WordNetLemmatizer()
```

```
import pandas as pd
import os
folder_path = "/content/drive/MyDrive/Colab Notebooks/teacher"  # replace with the path to your folder
files = []  # to store file names
contents = []  # to store file contents

for filename in os.listdir(folder_path):
    if filename.endswith('.txt'):  # consider only text files
        with open(os.path.join(folder_path, filename), 'r') as file:
            files.append(filename)
            contents.append(file.read())

df = pd.DataFrame({'file_name': files, 'content': contents})
df
```

	file_name	content
0	TeacherPoemsDeathOfATeacherPoembyCarstenThomse...	Thirteen years older\nbut hardly wiser\nI retu...
1	TeacherPoemsIfIBecomeATeacherPoembySnehaKhedka...	This is the first poem I have written.\n------...
2	TeacherPoemsFutureTeacherPoembyDyhanaraRios.txt	(1st place winner of Texas Career Association ...
3	TeacherPoemsMyTeacherAteMyHomeworkPoembyshaikh...	my teacher ate my homework,\nwhich i thought w...
4	TeacherPoemsTeacherManPoembyarybolanos.txt	The Teacher Man teaches kids from all over the...
...	...	...
95	TeacherPoemsYouSoundLikeTheTeacherInPeanutsPoe...	You want someone to hear you\nRun your mouth\n...
96	TeacherPoemsYouAreTheOneAndOnlyTeacherPoembyal...	i am taught well,\nnow my hands won't swell,\n...
97	TeacherPoemsTheMarkAGoodTeacherMakesPoembyJuli...	That little cherub sitting there\nWith eager

```
import nltk
from nltk.stem import WordNetLemmatizer
lemma = WordNetLemmatizer()
nltk.download('stopwords')
from nltk.corpus import stopwords

df['content']=df['content'].str.lower()
df['content'].head()

STOPWORDS = set(stopwords.words('english'))
```

```
def cleaning_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])
df['content'] = df['content'].apply(lambda text: cleaning_stopwords(text))
df['content'].head()
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
0    thirteen years older hardly wiser return past ...
1    first poem written. ---------- become te...
2    (1st place winner texas career association poe...
3    teacher ate homework, thought rather odd. snif...
4    teacher man teaches kids world teacher man tea...
Name: content, dtype: object
```

```
import string
english_punctuations = string.punctuation
punctuations_list = english_punctuations
def cleaning_punctuations(text):
    translator = str.maketrans('', '', punctuations_list)
    return text.translate(translator)
df['content']= df['content'].apply(lambda x: cleaning_punctuations(x))
df['content'].head()
```

```
0    thirteen years older hardly wiser return past ...
1    first poem written  become teacher respect chi...
2    1st place winner texas career association poet...
3    teacher ate homework thought rather odd sniffe...
4    teacher man teaches kids world teacher man tea...
Name: content, dtype: object
```

```
import re
def cleaning_repeating_char(text):
    return re.sub(r'(.)1+', r'1', text)
df['content'] = df['content'].apply(lambda x: cleaning_repeating_char(x))
df['content'].head()
```

```
0    thirteen years older hardly wiser return past ...
1    first poem written  become teacher respect chi...
2    1st place winner texas career association poet...
3    teacher ate homework thought rather odd sniffe...
4    teacher man teaches kids world teacher man tea...
Name: content, dtype: object
```

```
def cleaning_numbers(data):
    return re.sub('[0-9]+', ' ', data)
df['content'] = df['content'].apply(lambda x: cleaning_numbers(x))
df['content'].head()
```

```
0    thirteen years older hardly wiser return past ...
1    first poem written  become teacher respect chi...
2     st place winner texas career association poet...
3    teacher ate homework thought rather odd sniffe...
4    teacher man teaches kids world teacher man tea...
Name: content, dtype: object
```

```
def transform_text(text):
    return ' '.join([word for word in text.split() if len(word) > 2])
df['content'] = df['content'].apply(lambda x: transform_text(x))
df['content'].head()
```

```
0    thirteen years older hardly wiser return past ...
1    first poem written become teacher respect chil...
2    place winner texas career association poetry c...
3    teacher ate homework thought rather odd sniffe...
4    teacher man teaches kids world teacher man tea...
Name: content, dtype: object
```

```
import nltk
st = nltk.PorterStemmer()
def stemming_on_text(data):
    text = [st.stem(word) for word in data]
    return data
df['content']= df['content'].apply(lambda x: stemming_on_text(x))
df['content'].head()
```

```
0    thirteen years older hardly wiser return past ...
1    first poem written become teacher respect chil...
2    place winner texas career association poetry c...
3    teacher ate homework thought rather odd sniffe...
4    teacher man teaches kids world teacher man tea...
Name: content, dtype: object
```

```
import nltk
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
True
```

```
nltk.download('omw-1.4')
lm = nltk.WordNetLemmatizer()
def lemmatizer_on_text(data):
    text = [lm.lemmatize(word) for word in data]
    return data
df['content'] = df['content'].apply(lambda x: lemmatizer_on_text(x))
df['content'].head()
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
0    thirteen years older hardly wiser return past ...
1    first poem written become teacher respect chil...
2    place winner texas career association poetry c...
3    teacher ate homework thought rather odd sniffe...
4    teacher man teaches kids world teacher man tea...
Name: content, dtype: object
```

```
from nltk.tokenize import TweetTokenizer
df2=df.iloc[:,0:1]
tt = TweetTokenizer()
df2['content']=df['content'].apply(tt.tokenize)
df2['content']
```

```
0     [thirteen, years, older, hardly, wiser, return...
1     [first, poem, written, become, teacher, respec...
2     [place, winner, texas, career, association, po...
3     [teacher, ate, homework, thought, rather, odd,...
4     [teacher, man, teaches, kids, world, teacher, ...
                            ...
95    [want, someone, hear, run, mouth, one, wants, ...
96    [taught, well, hands, swell, cause, learned, d...
97    [little, cherub, sitting, eager, eyes, expecta...
98    [peach, i, ’, ve, eaten, graced, blessed, taug...
99    [teacher, must, know, plato, well, said, years...
Name: content, Length: 100, dtype: object
```

Generate Inverted Index (variation in data structures)

```
def create_inverted_index(tokenized_tweets):
    inverted_index = dict()
    for document_idx in range(len(tokenized_tweets)):
        for word in tokenized_tweets[document_idx]:
            if word not in inverted_index:
                inverted_index[word] = list()
            inverted_index[word].append(document_idx)

    for key in inverted_index:
        inverted_index[key] = list(set(inverted_index[key]))

    return inverted_index

inverted_index = create_inverted_index(df2['content'])
```

```
inverted_index
```

```
{'thirteen': [0, 66, 12],
 'years': [0, 65, 99, 36, 41, 45, 48, 24, 56, 91],
 'older': [0, 91],
 'hardly': [0],
 'wiser': [0, 82],
 'return': [0, 96, 40, 41, 82],
 'past': [0, 80, 11, 46],
 'find': [0, 67, 4, 71, 17, 50, 81, 23, 56, 91, 63],
 'six': [0, 41],
 'months': [0],
 'dead': [0, 38],
 'still': [0, 2, 99, 7, 41, 11, 15, 17, 82, 52, 23, 63],
 'alive': [0, 65, 56, 38],
 'head': [0, 38, 46, 82, 21],
 'memory': [0],
 'infamous': [0],
```

```
 'three': [0, 82, 90, 31],
 'days': [0, 14, 48, 82, 85, 56, 60],
 'nights': [0],
 'sustaining': [0],
 'sunflowerseeds': [0],
 'moonshine': [0],
 'pissed': [0],
 'hours': [0, 82],
 'away': [0, 70, 40, 74, 82, 52, 84, 85, 57],
 'till': [0, 25],
 'gave': [0, 24, 40, 3],
 'went': [0, 3, 75, 46, 81, 82, 60, 93],
 'home': [0, 68, 5, 38, 85, 29, 63],
 'alone': [0, 81, 82, 86, 26],
 'biggest': [0],
 'hangovers': [0],
 'ever': [0, 35, 41, 91, 45, 77, 15, 47, 82, 52, 85, 23, 88, 89, 59, 93, 94],
 'died': [0, 82],
 'alcoholic': [0],
 'fourtyfive': [0],
 'wonder': [0, 78, 82, 85, 93],
 'caurse': [0],
 'excessive': [0],
 'indifferent': [0, 40],
 'recklessness': [0],
 'someone': [0, 65, 66, 4, 7, 12, 26, 91, 95],
 'break': [0, 8, 52, 49],
 'heart': [0, 33, 34, 67, 96, 71, 40, 46, 81, 19, 52, 85, 93, 25, 60, 63, 57],
 'loose': [0],
 'faith': [0, 2, 81, 82, 85],
 'life': [0,
  2,
  4,
  5,
  7,
  9,
  11,
  13,
  16,
  19,
  22,
  24,
```

Boolean Query

```
def parse_query(infix_tokens):

    precedence = {}
    precedence['NOT'] = 3
    precedence['AND'] = 2
    precedence['OR'] = 1
    precedence['('] = 0
    precedence[')'] = 0

    output = []
    operator_stack = []

    for token in infix_tokens:
        if (token == '('):
            operator_stack.append(token)

        elif (token == ')'):
            operator = operator_stack.pop()
            while operator != '(':
                output.append(operator)
                operator = operator_stack.pop()

        elif (token in precedence):
            if (operator_stack):
                current_operator = operator_stack[-1]
                while (operator_stack and precedence[current_operator] > precedence[token]):
                    output.append(operator_stack.pop())
                    if (operator_stack):
                        current_operator = operator_stack[-1]
            operator_stack.append(token) # add token to stack
        else:
            output.append(token.lower())

    while (operator_stack):
        output.append(operator_stack.pop())
```

```
    return output


def boolean_query(query, inverted_index):
    query = query.strip()
    query_tokens = query.split()
    boolean_query = parse_query(query_tokens)

    result_stack = list()
    for idx, token in enumerate(boolean_query):
        if token not in ["AND", "NOT", "OR"]:
            result = set(inverted_index[token])
        else:
            if token in ['AND', 'OR']:
                right_operand = result_stack.pop()
                left_operand = result_stack.pop()

                if token == 'AND':
                    operation = set.intersection
                else:
                    operation = set.union

                result = operation(left_operand, right_operand)

            else:
                operand = result_stack.pop()
                complement_document_ids = inverted_index[boolean_query[idx-1]]
                result = list()
                for word in inverted_index:
                    result.extend([_id for _id in inverted_index[word] if _id not in complement_document_ids])
                result = set(result)

        result_stack.append(result)

    return result_stack.pop()


document_ids = boolean_query("sitting AND class", inverted_index)
print(f"Document IDs: {document_ids}")
```

```
Document IDs: {58, 11}
```

```
document_ids = boolean_query("sitting AND class OR quickly", inverted_index)
print(f"Document IDs: {document_ids}")
```

```
Document IDs: {59, 58, 11}
```

3. handling wild card and phrase queries

```
def construct_positional_posting_list(tokenized_corpus):
    positional_index = dict()
    for tweet_id, tweet in enumerate(tokenized_corpus):
        for token_id, token in enumerate(tweet):
            if token not in positional_index:
                positional_index[token] = dict()
            if tweet_id not in positional_index[token]:
                positional_index[token][tweet_id] = list()
            positional_index[token][tweet_id].append(token_id)

    for token in positional_index:
        for tweet_id in positional_index[token]:
            positional_index[token][tweet_id] = sorted(positional_index[token][tweet_id])
        items = list(positional_index[token].items())
        items.sort(key=lambda x: x[0])
        for k, v in items:
            positional_index[token][k] = v

    return positional_index


positional_index = construct_positional_posting_list(df2['content'])


for key in list(positional_index.keys())[:10]:
    print(f"Word: {key}\nTweet & Token Indices: {positional_index[key]}\n")
```

```
Word: thirteen
Tweet & Token Indices: {0: [0], 12: [26], 66: [26]}

Word: years
Tweet & Token Indices: {0: [1], 24: [64], 36: [34], 41: [147, 156], 45: [36], 48: [5], 56: [1, 19, 75, 88], 65: [30], 91: [159], 99

Word: older
Tweet & Token Indices: {0: [2], 91: [178]}

Word: hardly
Tweet & Token Indices: {0: [3]}

Word: wiser
Tweet & Token Indices: {0: [4], 82: [247]}

Word: return
Tweet & Token Indices: {0: [5], 40: [118], 41: [128], 82: [701], 96: [37]}

Word: past
Tweet & Token Indices: {0: [6], 11: [62, 95, 115], 46: [65], 80: [13]}

Word: find
Tweet & Token Indices: {0: [7], 4: [149], 17: [9], 23: [25], 50: [24], 56: [101], 63: [96], 67: [63], 71: [42], 81: [21], 91: [128]

Word: six
Tweet & Token Indices: {0: [8], 41: [94]}

Word: months
Tweet & Token Indices: {0: [9]}
```

```
def construct_biword_positional_posting_list(tokenized_corpus):
    biword_positional_index = dict()
    for tweet_id, tweet in enumerate(tokenized_corpus):
        for token_id, token in enumerate(tweet[:-1]): # iterate till second last token
            biword = token + ' ' + tweet[token_id + 1]
            if biword not in biword_positional_index:
                biword_positional_index[biword] = dict()
            if tweet_id not in biword_positional_index[biword]:
                biword_positional_index[biword][tweet_id] = list()
            biword_positional_index[biword][tweet_id].append(token_id)

    for biword in biword_positional_index:
        for tweet_id in biword_positional_index[biword]:
            biword_positional_index[biword][tweet_id] = sorted(biword_positional_index[biword][tweet_id])
        items = list(biword_positional_index[biword].items())
        items.sort(key=lambda x: x[0])
        for k, v in items:
            biword_positional_index[biword][k] = v

    return biword_positional_index


biword_indexing = construct_biword_positional_posting_list(df2['content'])


biword_indexing
```

```
{'thirteen years': {0: [0]},
 'years older': {0: [1]},
 'older hardly': {0: [2]},
 'hardly wiser': {0: [3]},
 'wiser return': {0: [4]},
 'return past': {0: [5]},
 'past find': {0: [6]},
 'find six': {0: [7]},
 'six months': {0: [8]},
 'months dead': {0: [9]},
 'dead still': {0: [10]},
 'still alive': {0: [11]},
 'alive head': {0: [12]},
 'head memory': {0: [13]},
 'memory infamous': {0: [14]},
 'infamous three': {0: [15]},
 'three days': {0: [16], 82: [672, 694]},
 'days nights': {0: [17]},
 'nights sustaining': {0: [18]},
 'sustaining sunflowerseeds': {0: [19]},
 'sunflowerseeds moonshine': {0: [20]},
 'moonshine pissed': {0: [21]},
 'pissed hours': {0: [22]},
 'hours days': {0: [23]},
 'days away': {0: [24]},
 'away till': {0: [25]},
```

```
 'till gave': {0: [26]},
 'gave went': {0: [27]},
 'went home': {0: [28]},
 'home alone': {0: [29]},
 'alone biggest': {0: [30]},
 'biggest hangovers': {0: [31]},
 'hangovers ever': {0: [32]},
 'ever died': {0: [33]},
 'died alcoholic': {0: [34]},
 'alcoholic fourtyfive': {0: [35]},
 'fourtyfive wonder': {0: [36]},
 'wonder caurse': {0: [37]},
 'caurse excessive': {0: [38]},
 'excessive indifferent': {0: [39]},
 'indifferent recklessness': {0: [40]},
 'recklessness someone': {0: [41]},
 'someone break': {0: [42]},
 'break heart': {0: [43]},
 'heart loose': {0: [44]},
 'loose faith': {0: [45]},
 'faith life': {0: [46]},
 'life yourself': {0: [47]},
 'yourself toasted': {0: [48]},
 'toasted night': {0: [49]},
 'night day': {0: [50]},
 'day black': {0: [51]},
 'black white': {0: [52]},
 'white life': {0: [53]},
 'life death': {0: [54]},
 'death ones': {0: [55]},
 'ones you': {0: [56]},
```

```
for key in list(biword_indexing.keys())[:10]:
    print(f"Word: {key}\nTweet & Token Indices: {biword_indexing[key]}\n")
```

```
 Word: thirteen years
 Tweet & Token Indices: {0: [0]}

 Word: years older
 Tweet & Token Indices: {0: [1]}

 Word: older hardly
 Tweet & Token Indices: {0: [2]}

 Word: hardly wiser
 Tweet & Token Indices: {0: [3]}

 Word: wiser return
 Tweet & Token Indices: {0: [4]}

 Word: return past
 Tweet & Token Indices: {0: [5]}

 Word: past find
 Tweet & Token Indices: {0: [6]}

 Word: find six
 Tweet & Token Indices: {0: [7]}

 Word: six months
 Tweet & Token Indices: {0: [8]}

 Word: months dead
 Tweet & Token Indices: {0: [9]}
```

```
def positional_intersect(p1, p2, K):
    answer = list()
    i = 0
    j = 0
    while i < len(p1) and j < len(p2):
        document_id_p1 = list(p1.keys())[i]
        document_id_p2 = list(p2.keys())[j]

        if document_id_p1 == document_id_p2:
            l = list()
            pp1 = p1[document_id_p1]
            pp2 = p2[document_id_p2]

            k = 0
            while k < len(pp1):
                m = 0
                while m < len(pp2):
                    distance = pp2[m] - pp1[k]
                    if distance == K:
                        l.append(m)
```

```
                    m += 1

                for ps in l:
                    distance = (pp2[ps] - pp1[k])
                    if distance != K:
                        l.remove(ps)

                for ps in l:
                    answer.append((document_id_p1, pp1[k], pp2[ps]))

                k += 1

            i += 1
            j += 1

        elif document_id_p1 < document_id_p2:
            i += 1
        else:
            j += 1

    return answer
```

K gram

```
def generate_k_grams(term, k):
    """
    Generate K-grams for a given term.
    """
    term = "#" + term + "#"  # Add '#' at the start and end of the term
    k_grams = [term[i:i+k] for i in range(len(term)-k+1)]
    return k_grams


def construct_kgram_biword_positional_posting_list(tokenized_corpus, k):
    biword_positional_index = dict()
    kgram_index = dict()
    for tweet_id, tweet in enumerate(tokenized_corpus):
        for token_id, token in enumerate(tweet[:-1]): # iterate till second last token
            biword = token + ' ' + tweet[token_id + 1]
            k_grams = generate_k_grams(biword, k)
            for kg in k_grams:
                if kg not in kgram_index:
                    kgram_index[kg] = set()
                kgram_index[kg].add(biword)
                if biword not in biword_positional_index:
                    biword_positional_index[biword] = dict()
                if tweet_id not in biword_positional_index[biword]:
                    biword_positional_index[biword][tweet_id] = list()
                biword_positional_index[biword][tweet_id].append(token_id)

    for biword in biword_positional_index:
        for tweet_id in biword_positional_index[biword]:
            biword_positional_index[biword][tweet_id] = sorted(biword_positional_index[biword][tweet_id])
        items = list(biword_positional_index[biword].items())
        items.sort(key=lambda x: x[0])
        for k, v in items:
            biword_positional_index[biword][k] = v
    return biword_positional_index, kgram_index


def retrieve_wildcard_postings(wildcard_query, biword_positional_index, kgram_index, k):
    """
    Retrieve postings for a wildcard query using K-gram index.
    """
    k_grams = generate_k_grams(wildcard_query, k)
    candidate_biwords = set()
    for kg in k_grams:
        if kg in kgram_index.keys():
            candidate_biwords.update(kgram_index[kg])
    result = dict()
    for biword in candidate_biwords:
        if wildcard_query in biword:
            result[biword] = biword_positional_index[biword]
    return result
```

```
corpus = list(df2['content'])
biword_index, kgram_index = construct_kgram_biword_positional_posting_list(corpus, k=3)

query = "student"
results = retrieve_wildcard_postings(query, biword_index, kgram_index, k=3)
print(results)
```

```
{'helps student': {51: [20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20]}, 'students fight': {20: [66, 66, 66, 66, 66, 66, 66, 6
```

```
def isMatchRecursive(s, p ,p1, p2):
    if p2 == len(p):
        return len(s) == p1
    if p[p2] != '*':
        if p1 < len(s) and (s[p1] == p[p2] or p[p2] == '?'):
            return isMatchRecursive(s, p,p1+1,p2+1)
        else:
            return False
    else:
        while p2< len(p)-1 and p[p2+1] == '*':
            p2 += 1
        p1 -= 1

        while p1< len(s):
            if isMatchRecursive(s,p,p1+1, p2+1):
                return True

            p1 += 1

    return False

def isMatch(s, p):
    return isMatchRecursive(s, p, 0 , 0)


pattern=input("Enter pattern for wild card: ")
for i in inverted_index:
    #print(i)
    if(isMatch(i,pattern)):
        print(i,"prensent in document", inverted_index[i])
```

```
Enter pattern for wild card: te*er
teacher prensent in document [0, 1, 2, 3, 4, 6, 7, 8, 9, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 25, 26, 27, 28, 29, 30, 32, 33
teaher prensent in document [64]
```

4. Retrieve relevant text using similarity index

```
import spacy

# load a pre-trained model
nlp = spacy.load("en_core_web_md")

# define the text you want to compare against
query = "I am looking for a restaurant in New York City."

# define a list of candidate texts to compare against
str1=""
candidates= df2["content"]
for candidate in candidates:
  for i in candidate:
    str1=str1+i+" "
# calculate the similarity scores between the query and each candidate
scores = []
for candidate in candidates:
    doc = nlp(str1)
    score = doc.similarity(nlp(query))
    scores.append(score)

# find the candidate with the highest similarity score
index = scores.index(max(scores))
most_similar_text = candidates[index]

print("Most similar text:", most_similar_text)
```

```
Most similar text: ['thirteen', 'years', 'older', 'hardly', 'wiser', 'return', 'past', 'find', 'six', 'months', 'dead', 'still', 'a
```

5. Ranked Retrieval using tf-idf

```
DF = {}
for word in inverted_index.keys():
    DF[word] = len ([doc for doc in inverted_index[word]])

total_vocab_size = len(DF)
print(total_vocab_size)
```

```
2979
```

```
from tqdm import tqdm

from collections import Counter

tf_idf = {} # Our data structure to store Tf-Idf weights

N = len(df2['content'])

for doc_id, tokens in tqdm(df2['content'].items()):

    counter = Counter(tokens)
    words_count = len(tokens)

    for token in np.unique(tokens):

        # Calculate Tf
        tf = counter[token] # Counter returns a tuple with each terms counts
        tf = 1+np.log(tf)

        # Calculate Idf
        if token in DF:
            df = DF[token]
        else:
            df = 0
        idf = np.log((N+1)/(df+1))

        # Calculate Tf-idf
        tf_idf[doc_id, token] = tf*idf
```

```
100it [00:00, 843.57it/s]
```

```
import unicodedata
from nltk.corpus import stopwords
STOP_WORDS = set(stopwords.words('english'))
WORD_MIN_LENGTH = 2 ## we'll drop all tokens with less than this size
def strip_accents(text):
    """Strip accents and punctuation from text.
    For instance: strip_accents("João e Maria, não entrem!")
    will return "Joao e Maria  nao entrem "

    Parameters:
    text (str): Input text

    Returns:
    str: text without accents and punctuation

   """
    nfkd = unicodedata.normalize('NFKD', text)
    newText = u"".join([c for c in nfkd if not unicodedata.combining(c)])
    return re.sub('[^a-zA-Z0-9 \\\']', ' ', newText)

def tokenize_text(text):
    """Make all necessary preprocessing of text: strip accents and punctuation,
    remove \n, tokenize our text, convert to lower case, remove stop words and
    words with less than 2 chars.

    Parameters:
    text (str): Input text

    Returns:
    str: cleaned tokenized text

   """
    text = strip_accents(text)
    text = re.sub(re.compile('\n'),' ',text)
    words = word_tokenize(text)
```

```
    words = [word.lower() for word in words]
    words = [word for word in words if word not in STOP_WORDS and len(word) >= WORD_MIN_LENGTH]
    return words

def ranked_search(k, tf_idf_index, file_names, query):
    """Run ranked query search using tf-idf model.

    Parameters:
    k (int): number of results to return
    tf_idf_index (dict): Data Structure storing Tf-Idf weights to each
                         pair of (term,doc_id)
    file_names (list): List with names of files (books)
    query (txt): Query text

    Returns:
    Top-k names of books that matchs the query.

   """
    tokens = tokenize_text(query)
    query_weights = {}
    for doc_id, token in tf_idf:
        if token in tokens:
            query_weights[doc_id] = query_weights.get(doc_id, 0) + tf_idf_index[doc_id, token]

    query_weights = sorted(query_weights.items(), key=lambda x: x[1], reverse=True)
    results = []
    for i in query_weights[:k]:
        results.append(file_names[i[0]])

    return results
print(ranked_search(10, tf_idf, df2['content'], "teacher sitting in a class"))
```

```
[['i', '’', 'm', 'sitting', 'class', 'eyes', 'close', 'hope', 'teacher', 'don', '’', 't', 'see', 'hope', 'don', '’', 't', 'see', 'b
```

Semantic matching

```
from nltk.corpus import wordnet as wn
from nltk.tokenize import word_tokenize
def calculate_similarity(word1, word2):
    synsets1 = wn.synsets(word1)
    synsets2 = wn.synsets(word2)
    if not synsets1 or not synsets2:
        return 0.0
    max_sim = -1
    for synset1 in synsets1:
        for synset2 in synsets2:
            sim = wn.path_similarity(synset1, synset2)
            if sim is not None and sim > max_sim:
                max_sim = sim
    return max_sim

def semantic_matching(query):
    documents = inverted_index
    scores = []
    query_tokens = word_tokenize(query)
    for document in documents:
        doc_tokens = word_tokenize(document)
        similarity_score = 0.0
        for query_token in query_tokens:
            max_sim = -1
            for doc_token in doc_tokens:
                sim = calculate_similarity(query_token, doc_token)
                if sim > max_sim:
                    max_sim = sim
            similarity_score += max_sim
        scores.append((document, similarity_score / len(query_tokens)))
    return scores

x=input("Enter query: ")
semantic_matching(x)
```

```
Enter query: teacher
[('i', 0.09090909090909091),
 ('’', 0.0),
 ('m', 0.09090909090909091),
 ('sitting', 0.09090909090909091),
 ('class', 0.09090909090909091),
 ('eyes', 0.125),
 ('close', 0.09090909090909091),
```

```
('hope', 0.16666666666666666),
('teacher', 1.0),
('don', 0.5),
('t', 0.07142857142857142),
('see', 0.09090909090909091),
('bored', 0.09090909090909091),
('mind', 0.14285714285714285),
('falling', 0.09090909090909091),
('behind', 0.09090909090909091),
('cannot', 0.09090909090909091),
('concentrate', 0.1111111111111111),
('came', 0.09090909090909091),
('late', 0.09090909090909091),
('although', 0.0),
('time', 0.1),
('body', 0.1),
('mean', 0.09090909090909091),
('still', 0.09090909090909091),
('outside', 0.09090909090909091),
('trying', 0.09090909090909091),
('hide', 0.09090909090909091),
('haven', 0.08333333333333333),
('used', 0.09090909090909091),
('yet', 0.09090909090909091),
('retain', 0.08333333333333333),
('already', 0.09090909090909091),
('forget', 0.09090909090909091),
('walks', 0.08333333333333333),
('over', 0.2),
('mine', 0.09090909090909091),
('smile', 0.07692307692307693),
('gentle', 0.09090909090909091),
('one', 0.16666666666666666),
('that', 0.0),
('puts', 0.08333333333333333),
('hand', 0.125),
('sets', 0.1111111111111111),
('kind', 0.2),
('free', 0.09090909090909091),
('learning', 0.14285714285714285),
('past', 0.09090909090909091),
('history', 0.14285714285714285),
('learn', 0.09090909090909091),
('mistakes', 0.16666666666666666),
('old', 0.09090909090909091),
('people', 0.1),
('different', 0.09090909090909091),
('bold', 0.09090909090909091),
('stood', 0.08333333333333333),
('oppression', 0.07692307692307693),
```