**Τμήμα Μηχανικών Η/Υ & Πληροφορικής**

**Πανεπιστήμιο Πατρών**

Ανάκτηση Πληροφορίας

**Εργαστηριακή Άσκηση**

**Χειμερινό Εξάμηνο 2023**

**Νικόλαος Βούλγαρης, 1084626,** [**st1084626@ceid.upatras.gr**](mailto:st1084626@ceid.upatras.gr)

**Όλα τα αρχεία υπάρχουν και στο** [**https://github.com/NickVoulg02/Information-Retrieval**](https://github.com/NickVoulg02/Information-Retrieval)**. Από εκεί μπορείτε να εκτελέσετε και το colbert\_test\_link.ipynb αρχείο για το colBERT. Στο παράρτημα παρατίθεται ο κώδικας από όλα τα .py αρχεία.**

# **Ερώτημα 1 - Ανάγνωση και Επεξεργασία της Συλλογής**

Η δομή ενός ανεστραμμένου αρχείου αποτελείται από το λεξικό και τις εμφανίσεις.

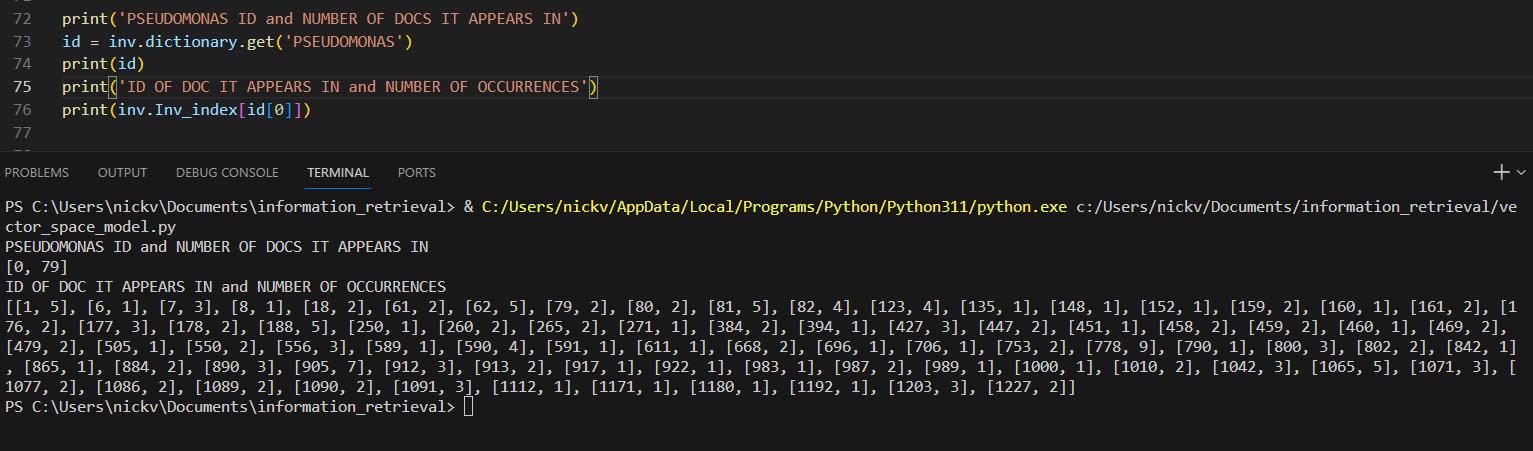
Το λεξικό αποτελείται από όλες τις μοναδικές λέξεις που εμφανίζονται στη συλλογή κειμένων. Προκειμένου να μειώσουμε τους χώρους της δομής ευρετηριοδότησης εξαλείφουμε τα stopwords. Στη δική μου υλοποίηση εξάλειψα και τους αριθμούς που εμφανίζονται στα κείμενα και μείωσα τις εγγραφές του λεξικού κατά 1000. Το λεξικό περιέχει το αναγνωριστικό της κάθε λέξης και ένα συνολικό αριθμό κειμένων που εμφανίζεται η λέξη.

Το ανεστραμμένο αρχείο αντιστοιχίζει μια λίστα εγγραφών σε κάθε λέξη. Οι εγγραφές αποτελούνται από το αναγνωριστικό του κείμενου που περιέχει την λέξη και τον αριθμό εμφανίσεων της λέξης στο συγκεκριμένο κείμενο.

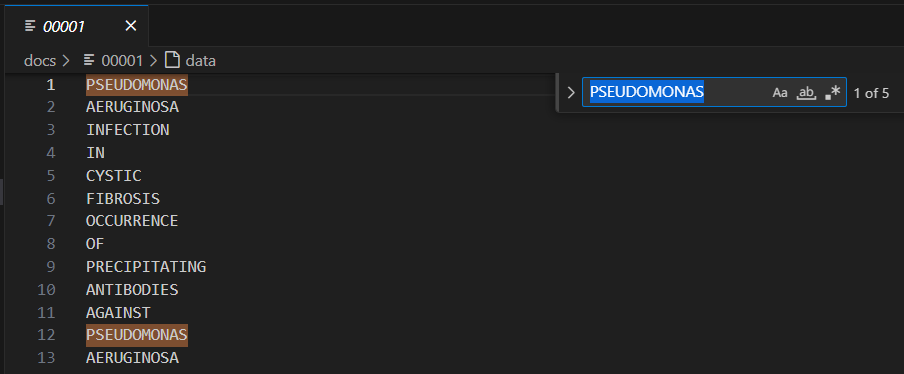
Στο κώδικα υλοποιείται με την class InvertedIndex, που έχει ένα dictionary τύπου python dictionary, ένα dic\_id ακέραιο και μια Inv\_index λίστα. Ανοίγουμε όλα τα κείμενα του υποφάκελου docs και για καθένα εκτελείται η ακόλουθη διαδικασία. Σπάμε τις γραμμές του με χρήση splitlnes() συνάρτησης, χρησιμοποιούμε το όνομα του αρχείου ως id και κρατάμε σε file\_len και number\_of\_docs μεταβλητές για το μέγεθος του αρχείου και τον αριθμό των αρχείων αντίστοιχα ( μεταβλητές που θα αξιοποιηθούν αργότερα ).

Καλείται Inverted\_Index της κλάσης InvertedIndex. Κάθε λέξη αντιμετωπίζεται σαν token. Αν αποτελεί αριθμό ή αν ανήκει στη stopwords.words('english') ττης ntlk βιβλιοθήκης δεν λαμβάνεται υπόψιν. Κρατάμε τον αριθμό των token μετά το φιλτράρισμα που εμφανίζονται σε κάθε κείμενο στη μεταβλητή length, που θα χρησιμοποιηθεί αργότερα. Αν το token δεν ανήκει στο dictionary, προστίθεται εκεί και προστίθεται ένας πίνακας με το id και τον αριθμό 1, για τις εμφανίσεις στο κείμενο μέχρι τώρα, στο Inv\_index. Αν ανήκει στο dictionary και πρόκειται για το ίδιο κείμενο που έχει εμφανιστεί ήδη τουλάχιστον μια φορά, απλά αυξάνουμε τον αριθμό εμφανίσεων της λέξης στο συγκεκριμένο κείμενο. Αλλιώς εισάγουμε στην εγγραφή για το συγκεκριμένο token άλλο ένα πίνακα 2 στοιχείων.

Ας δείξουμε ένα παράδειγμα για την πρώτη λέξη του πρώτου κειμένου, PSEUDOMONAS.



Ελέγχουμε το πρώτο και το έκτο κείμενο σύμφωνα με τα αποτελέσματα του Inv\_index.



6 εμφανίσεις στο πρώτο κείμενο

A screenshot of a computer

Description automatically generated

1 στο έκτο

Μετά το τέλος της συνάρτησης στο file\_len dictionary εισάγεται ένα στοιχείο με id αυτή του τρέχοντος κειμένου και τιμή το length που επιστράφηκε από την Inverted\_Index()

# **Ερώτημα 2 - Υλοποίηση Vector Space μοντέλου**

Το vector space είναι ένα μοντέλο ανάκτησης πληροφορίας για αναπαράσταση των κειμένων ως διανύσματα, με την απόσταση μεταξύ τους να αναπαριστά το πόσο σχετικά είναι τα 2 κείμενα. Το βάρος των όρων σε κάθε κείμενο χρησιμοποιείται για τον βαθμό ομοιότητας. Τα βάρη υπολογίζονται με χρήση της TF-IDF. H Term Frequency υπολογίζει τον αριθμό εμφάνισης του όρου σε κάθε κείμενο. Η Inverse Document Frequency υπολογίζει τον αριθμό εμφάνισης του όρου σε όλα τα κείμενα της συλλογής, υπολογίζει ουσιαστικά τη σπανιότητα του.

Στην υλοποίηση μου τα βάρη idf υπολογίζονται με βάση τον τύπο απλής λογαριθμικής κανονικοποίησης log(1+𝑁/𝑛𝑖), όπου N το πλήθος των κειμένων και ni ο αριθμός των κειμένων που εμφανίζεται το token.

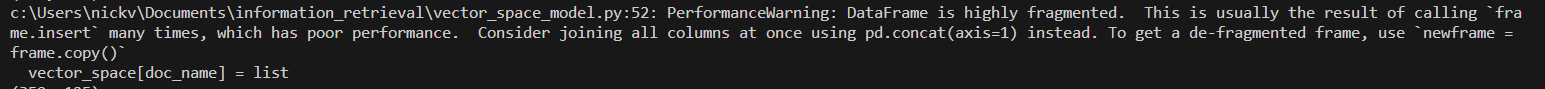
Αποθηκεύουμε τα αποτελέσματα στην idf\_values λίστα.

Τα βάρη tf υπολογίζονται με τον τύπο της απλής λογαριθμικής κανονικοποίησης log(1+fi,j), όπου fi,j το πηλίκο του αριθμού εμφανίσεων ενός token στο κείμενο προς το πλήθος των token μες στο κείμενο.

Σε κάθε κείμενο ελέγχουμε αν κάποιο token υπάρχει σε αυτό. Πρέπει να αναζητήσουμε ολόκληρη την εγγραφή στο Inv\_index, έτσι ώστε να βρούμε τον αριθμό εμφανίσεων για τις τιμές που ταυτίζονται με το όνομα του κειμένου που εξετάζουμε. Αν το βρούμε υπολογίζουμε το tf βάρος, το πολλαπλασιάζουμε με το αντίστοιχο idf και τα αποθηκεύουμε σε μια λίστα. Αν δεν εμφανίζεται το token στο συγκεκριμένο κείμενο αποθηκεύουμε μηδέν.

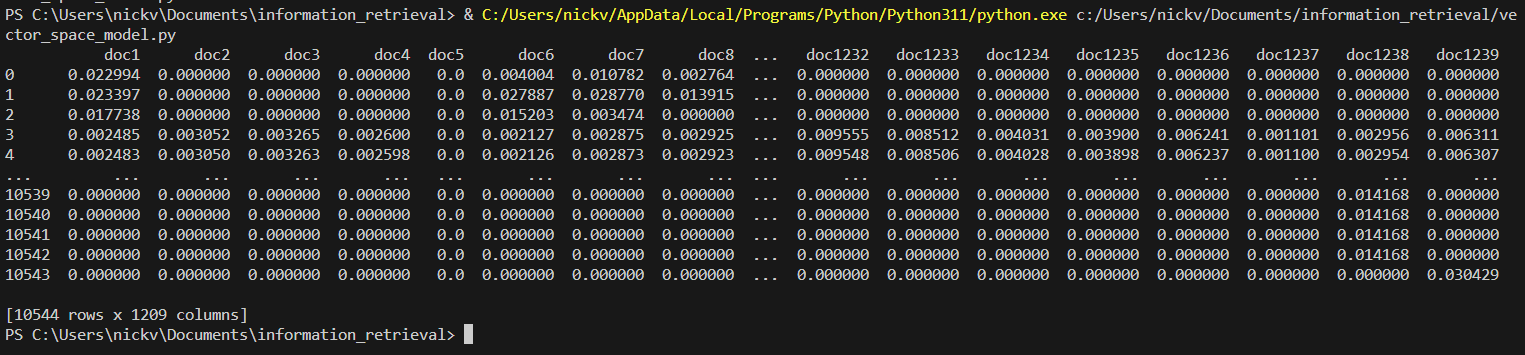
Για κάθε κείμενο η λίστα αποθηκεύεται σε ένα pandas.Dataframe ονόματι vector\_space. Στο τέλος εξάγεται σε ένα csv αρχείο, όπου κάθε στήλη αντιστοιχεί σε ένα document και κάθε γραμμή σε ένα token του dictionary.

H γραμμή simplefilter(action="ignore", category=pd.errors.PerformanceWarning) αφορά το παρακάτω error που προκαλούσε η γραμμή vector\_space[doc\_name] = list



Εφόσον σκοπός της εργαστηριακής άσκησης δεν είναι η απόδοση όσο αφορά το χρόνο δημιουργίας του vector space, επέλεξα να αγνοήσω τις προειδοποιήσεις. Ο κώδικας εξάλλου εκτελείται σε λογικά χρονικά πλαίσια.

Εδώ είναι τα αποτελέσματα και η τελική μορφή του vector space model:



Εξάγεται ως vector\_space2.csv

# **Ερώτημα 3 - Υλοποίηση colBERT μοντέλου**

Στο ColBERT (Contextualized Late interaction over BERT), τα Queries και Documents κωδικοποιούνται ξεχωριστά σε contextual embeddings χρησιμοποιώντας δύο διαφορετικά μοντέλα BERT, γλωσσικά μοντέλα που βασίζονται στην αρχιτεκτονική των transformer. Τα contextual embeddings είναι απλά διανύσµατα που παράγονται ως έξοδοι από τα µοντέλα BERT. Τα 2 σύνολα κωδικοποιήσεων (ένα σύνολο για το ερώτημα q και ένα άλλο σύνολο tokens για το έγγραφο d) επιτρέπεται να παρακολουθούν το ένα το άλλο και να υπολογίζουν μια βαθμολογία για το πόσο σχετικά είναι το ερώτημα με έγγραφο για κάθε ζεύγος. Το έγγραφο που επιτυγχάνει την υψηλότερη βαθμολογία συνάφειας για ένα ερώτημα λαμβάνει τη χαμηλότερη θέση και αντίστροφα. Με αυτόν τον τρόπο, κατατάσσουμε το σύνολο των εγγράφων.

Ένα προ-εκπαιδευμένο μητρώο ενσωμάτωσης χρησιμοποιείται για τη δημιουργία token για τα q και d.

Στην δική μου υλοποίηση χρησιμοποιώ το Google Colab Notebook, που αναφέρεται και στη εκφώνηση της άσκησης. Αφού εισαχθούν όλες οι κατάλληλες βιβλιοθήκες για το colBERT, εισάγω μέσω του προσωπικού μου GitHub repository 2 tsv αρχεία που αντιπροσωπεύουν τα αρχεία του φακέλου docs και των ερωτήσεων από το αρχείο Queries\_20. O κώδικας για την δημιουργία αυτών των αρχείων περιλαμβάνεται στο αρχείο preprocess.py. Για κάθε αρχείο του φακέλου το αποθηκεύουμε ολόκληρο μαζί με μια στήλη που κρατά το id του. Προσθέτουμε και ένα header. Για τις ερωτήσεις από το Queries\_20 κάνουμε παρόμοια διαδικασία αλλά δε συμπεριλαμβάνουμε header.

Δημιουργήθηκε collection Dataset της βιβλιοθήκης Hugging Face Transformers library μέσω pandas Dataframes που άνοιξε το doc\_col αρχείo και query Queries που άνοιξε το queries\_20.

A screenshot of a computer program

Description automatically generated

A black and white text

Description automatically generated with medium confidence

Υπολογίζουμε την colBERT αναπαράσταση του κάθε κειμένου και τα δεικτοδοτούμε. Ο indexer χρησιμοποιεί το colbert checkpoint που χρησιμοποιείται για να κάνουμε search. Ψάχνοντας τα αποτελέσματα για την πρώτη ερώτηση τα αποτελέσματα είναι τα εξής.

A screen shot of a computer screen

Description automatically generated

# **Ερώτημα 4 - Συγκρίσεις**

Όσο αφορά το vector space μοντέλο, για να εξετάσουμε την απόδοση του με βάσει μετρικές αξιολόγησης θα πρέπει να εισάγουμε διανύσματα για τις ερωτήσεις από το Queries\_20 αρχείο μέσα στο διανυσματικό χώρο. H διαδικασία είναι παρόμοια με αυτή που κάναμε για τα κείμενα του docs, το βάρος idf παραμένει το ίδιο για κάθε token που περιλαμβάνεται μέσα σε ένα query. Τα tf βάρη πρέπει να υπολογιστούν για την συχνότητα των token μέσα σε κάθε query και το τελικό αποτέλεσμα αποθηκεύεται στο query\_vec τύπου pandas.Dataframe.

Εφόσον έχουμε στο vector space διανύσματα για τα κείμενα και για τα ερωτήματα αρκεί να βρούμε την cosine similarity των διανυσμάτων, προκειμένου να βρούμε ποιο κείμενο είναι σχετικότερο με ποιο ερώτημα. Το cosine similarity 2 διανυσμάτων ισούται με το πηλίκο του εσωτερικού γινομένου και γινομένου των μέτρων τους. Στην υλοποίηση υπολογίζεται με χρήση των dot() και norm() συναρτήσεων της numpy βιβλιοθήκης και το αποτέλεσμα αποθηκεύεται στη retrieved\_docs λίστα για κάθε ερώτημα.

Εφόσον έχουμε μεθόδους που επιστρέφουν τα κείμενα σε μια διατεταγμένη ως προς την ομοιότητα λίστα, επέλεξα να χρησιμοποιήσω 2 μετρικές αξιολόγησης που είναι order aware και ελέγχουν αν τα σχετικά κείμενα είναι αυτά που κατατάσσονται υψηλότερα. Η Mean Average precision αξιολογεί κατά πόσον όλα τα στοιχεία που επιλέγονται από το μοντέλο ταυτίζονται με τα κείμενα που απορυθμίζονται στο Relevant\_20 και κατατάσσονται υψηλότερα ή όχι. Η Mean Reciprocal Rank αξιολογεί κατά πόσο επιστρέφει το καλύτερο σχετικό στοιχείο και θέλουμε αυτό το στοιχείο να βρίσκεται σε υψηλότερη θέση.

Ο τύπος για το Average Precision at k είναι

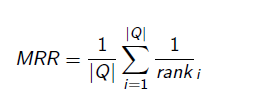
A math formula with symbols

Description automatically generated with medium confidence

Όπου GTP o αριθμός των σχετικών κειμένων, P@k το precision at k δηλαδή το πηλίκο του αριθμού σχετικών ανακτηθέντων κειμένων ως το προς k σημείο που έχουμε φτάσει στην κατάταξη και rel@k το relevance score για το κάθε κείμενο. Στην δική μας περίπτωση το relevance score είναι 1 αν το κείμενο είναι σχετικό, αλλιώς μηδέν. Αφού βρούμε το AP για κάθε κείμενο η μέση ακρίβεια σε πολλά ερωτήματα έχει τύπο

MAP = sum(AP(i))/Q.

Ο τύπος για το Mean Reciprocal Rank είναι



Όπου Q το πλήθος των queries και rank i η πρώτη θέση που βρίσκουμε κείμενο που είναι relevant μέσα στα κείμενα που επιστρέφει το μοντέλο.

Όσο αφορά το MAP στο Vector Space μοντέλο, για κάθε query υπολογίζουμε το cosine similarity ανάμεσα στο διάνυσμα αυτού με κάθε διάνυσμα κειμένου. Από τα αποτελέσματα κρατάμε τα πρώτα 100 για κάθε ερώτημα. Ελέγχουμε αν τα κείμενα που έχουν ανακτηθεί ανήκουν στα relevant που μας δίνονται. Τα αποτελέσματα αποθηκεύονται στη precision\_at\_k λίστα και αθροίζονται στο average\_precision. Τα average\_precision των κειμένων αποθηκεύονται στη avg\_pr\_list για να υπολογιστεί η mean\_average\_precision. Η διαδικασία είναι παρόμοια για το colBERT, χρησιμοποιούμε το passages\_ranked dictionary, όπου για κάθε id κειμένου κρατάμε το rank του στα top-k αποτελέσματα.

A screenshot of a computer

Description automatically generated

Αποτελέσματα MAP στο VSM

A screenshot of a computer program

Description automatically generated

*Αποτελέσματα MAP στο colBERT*

Παρακάτω φαίνονται 2 pyplot για σύγκριση των αποτελεσμάτων των 2 μεθόδων.

A graph with blue lines

Description automatically generatedA line graph with numbers and a line

Description automatically generated

Όσο αφορά το MRR στο Vector Space μοντέλο, για κάθε query κρατάμε τη θέση του πρώτου relevant κειμένου που ανακτήθηκε στη μεταβλητή mean\_rep\_rank.

A black screen with white text

Description automatically generated

MRR αποτελέσματα στο VSM

A screen shot of a computer

Description automatically generated

MRR στο colBERT

# Παρατηρούμε πως το Vector Space μοντέλο επιστρέφει καλύτερα αποτελέσματα σε σχέση με το colBERT, αλλά βρίσκονται σχετικά κοντά. Είναι πιθανό να υπερτερεί λόγω της μορφής των ερωτημάτων. Στηριζόμαστε στις εμφανίσεις των όρων μες στο κείμενο και όχι στην σημασιολογική σχέση όρων που έχουν κάποιο context. Αν το ζήτημα ήταν να επιστρέψουμε αποτελέσματα που θα ταιριάζουν και σημασιολογικά με τις ερωτήσεις το colBERT σίγουρα θα απέδιδε καλύτερα.

# Αναφορές

[Salton, G., Buckley, C., 1988. Term-weighting Approaches in Automatic Text Retrieval. Inf Process Manage 24, 513–523. https://doi.org/10.1016/0306-4573(88)90021-0](https://www.zotero.org/google-docs/?8hd24a)

[Salton, G., Wong, A., Yang, C.S., 1975. A Vector Space Model for Automatic Indexing. Commun ACM 18, 613–620. https://doi.org/10.1145/361219.361220](https://www.zotero.org/google-docs/?8hd24a)

Breaking Down Mean Average Precision (mAP), Ren Jie Tan, <https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52>

Mean reciprocal rank, <https://en.wikipedia.org/wiki/Mean_reciprocal_rank>

# Παράρτημα

Inverted\_index.py

import os

import nltk

from nltk.corpus import stopwords

#nltk.download('stopwords')

stopping = []

for x in stopwords.words('english'):        # A stop word is a commonly used word that a search engine has been programmed to ignore

    stopping.append(x.upper())

class InvertedIndex:

    def \_\_init\_\_(self):

        self.dictionary = {}                # a dictionary of all the words used in the articles, along with an id

                                            # and a number of total articles they appear in

        self.dic\_id = -1                    # used for the token ids in the dictionary

        self.Inv\_index = []                 # contains article id and number of occurences of token in said article

    def Tokenization(self, doc):

        words= []

        for x in range(len(doc)):

            if doc[x].isdigit() == False:                                   # removing every number from dictionary

                words.append(doc[x])

        return words

    def Inverted\_Index(self, id, doc):

        length=0                                                            # length variable only considers terms that aren't stopwords

        tokenlist = self.Tokenization(doc)                                  # a list of every word in the article

        for x in range(len(tokenlist)):

            if tokenlist[x] not in stopping:                                # checking for stopwords

                length+=1

                if self.dictionary.get(tokenlist[x]) == None:

                    self.dic\_id+=1

                    self.dictionary[tokenlist[x]] = [self.dic\_id, 1]

                    self.Inv\_index.append([[id , 1]])

                elif self.dictionary.get(tokenlist[x]) != None:

                    token\_id = self.dictionary.get(tokenlist[x])[0]

                    if self.Inv\_index[token\_id][-1][0] == id:               # checking if we're still in the same article by article id

                        temp1 = self.Inv\_index[token\_id][-1]

                        temp1[1]+=1

                    else:

                        self.dictionary.get(tokenlist[x])[1]+=1

                        self.Inv\_index[token\_id].append([id , 1])

        return length

def main(inv, file\_len, number\_of\_docs):

    cwd = os.getcwd()                                          # open working directory

    cwd = cwd + "\docs"

    #print(cwd)

    for filename in os.listdir(cwd):

         with open(os.path.join(cwd, filename), 'r') as f:     # open in readonly mode

             file = f.read().splitlines()

             id = int(filename)                                # id depends on filename

             number\_of\_docs+=1

             file\_len[id] = inv.Inverted\_Index(id, file)

    return number\_of\_docs

vector\_space\_model.py

import math

from warnings import simplefilter

import pandas as pd

import inverted\_index

simplefilter(action="ignore", category=pd.errors.PerformanceWarning)

def idf\_calculation(doc\_num, term\_dic):

    idf\_result = []

    for value in term\_dic.items():

        #print(value)                                                   #value[1][1] is the number of documents where the term appears

        idf\_result.append(math.log10((doc\_num / value[1][1]) + 1))

    return idf\_result

inv = inverted\_index.InvertedIndex()

file\_len = {}

number\_of\_docs = 0

number\_of\_docs = inverted\_index.main(inv, file\_len, number\_of\_docs)

idf\_values = idf\_calculation(number\_of\_docs, inv.dictionary)  # calculate idf values for every token (how rare it is)

#print(idf\_values)

vector\_space = pd.DataFrame()  # vector space consisting of tokens and documents

# filling the vector space

for doc in file\_len.items():

    #print(doc)

    list = []

    for token in inv.dictionary.items():

        detected = 0

        id = token[1][0]                                    # document id

        term = inv.Inv\_index[id]                            # Inv\_index record for the specific token

        for i in range(len(term)):                          # record for every doc the token appears in

            if term[i][0] == doc[0]:                        # checking if it appears in the doc we're currently checking

                # print(token[0])

                # print(doc[1])

                # print(term[i][1])                         # occurences in specific file

                x1 = math.log10((term[i][1] / doc[1])+1)    #logarithmically scaled frequency

                #print(x1)

                x2 = idf\_values[id]

                # print(x2)

                list.append(x1 \* x2)

                # print(x1\*x2)

                detected = 1

        if detected == 0:

            list.append(0)

    doc\_name = "doc" + str(doc[0])

    vector\_space[doc\_name] = list

print(vector\_space)

vector\_space.to\_csv('vector\_space2.csv', encoding='utf-8')

metrics.py

avg\_pr\_list = []

    length = len(query\_vec.columns[0:])

    for i in range(length):

        retrieved\_docs = cosine\_similarity(vector\_space, query\_vec.iloc[:,i])       #first column

        retrieved\_docs.sort(key=lambda ret:ret[1], reverse=True)

        retrieved\_docs\_filtered = retrieved\_docs[:100]

        # for x in range(len(retrieved\_docs\_filtered)):

        #     print(retrieved\_docs\_filtered[x][0])

        precision\_at\_k = []

        true\_positives = 0

        for doc in retrieved\_docs\_filtered:

            if(doc[0] in relevant\_docs[i]):

                true\_positives+=1

                precision\_at\_k.append(true\_positives/(retrieved\_docs\_filtered.index(doc)+1))

        #print(precision\_at\_k)

        average\_precision = 0

        for x in range(len(precision\_at\_k)):

                average\_precision += precision\_at\_k[x]

        average\_precision = average\_precision/len(relevant\_docs[i])

        print(average\_precision)

        avg\_pr\_list.append(average\_precision)

    mean\_average\_precision = sum(avg\_pr\_list)/length

    print("Mean Average Precision:" + str(mean\_average\_precision))

    plt.xticks(numpy.arange(len(avg\_pr\_list)), numpy.arange(1, len(avg\_pr\_list)+1))

    ypoints = numpy.array(avg\_pr\_list)

    plt.xlabel("Query ID")

    plt.ylabel("Average Precision for VSM")

    plt.plot(ypoints)

    plt.show()

# #Mean Reciprocal Rank

    # mean\_rep\_rank = 0

    # length = len(query\_vec.columns[0:])

    # list=[]

    # for i in range(length):

    #     retrieved\_docs = cosine\_similarity(vector\_space, query\_vec.iloc[:,i])       #first column

    #     retrieved\_docs.sort(key=lambda ret:ret[1], reverse=True)

    #     #retrieved\_docs\_filtered = retrieved\_docs[:250]

    #     #print(retrieved\_docs\_filtered)

    #     for doc in retrieved\_docs:

    #         if(doc[0] in relevant\_docs[i]):

    #             value = retrieved\_docs.index(doc)+1

    #             #print(value)

    #             break

    #     mean\_rep\_rank += 1/value

    #     list.append(mean\_rep\_rank)

    # mean\_rep\_rank = mean\_rep\_rank/length

    # print("Mean Reciprocal Rank:" + str(mean\_rep\_rank))

Preprocess.py

import os

import csv

cwd = os.getcwd()                                          # open working directory

docs = cwd + "\docs"

with open(cwd+"/doc\_col.tsv", 'wt', newline='', encoding='utf-8') as out\_file:

    tsv\_writer = csv.writer(out\_file, delimiter='\t')

    tsv\_writer.writerow(["doc\_id", "doc"])

    for filename in os.listdir(docs):

        with open(os.path.join(docs, filename), 'r') as f:     # open in readonly mode

            file = f.read().splitlines()

            passage = " "

            passage = passage.join(file)

            id = int(filename)

            tsv\_writer.writerow([id, passage])

with open(cwd+"/queries\_20.tsv", 'wt', newline='\n', encoding='utf-8') as out\_file:

    tsv\_writer = csv.writer(out\_file, delimiter='\t')

    with open(os.path.join(cwd, 'Queries\_20'), 'r') as f:     # open in readonly mode

            file = f.read().splitlines()

            id=0

            for x in file:

                 x = x.upper()

                 tsv\_writer.writerow([id, x])

                 id+=1