Group : Ankitha Kumari Moodukudru, Manisha Sharma, Rishabh Agrawal

Libraries

In [44]:
```
import requests
import numpy as np
import pandas as pd
from bs4 import BeautifulSoup as bs
from urllib.request import urlopen as uReq
import re
import time
from collections import defaultdict
import operator
import math
import nltk
import os
from os import listdir
import matplotlib.pyplot as plt
import string
from collections import OrderedDict
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from difflib import get_close_matches
```

In [2]:
```
class color:
   PURPLE = '\033[95m'
   CYAN = '\033[96m'
   DARKCYAN = '\033[36m'
   BLUE = '\033[94m'
   GREEN = '\033[92m'
   YELLOW = '\033[93m'
   RED = '\033[91m'
   BOLD = '\033[1m'
   UNDERLINE = '\033[4m'
   GO = '\033[0;30;43m'
   END = '\033[0m'
```

In [3]:
```
remove = string.punctuation
remove = remove.replace("-", "")
pattern = r"[{}]".format(remove)
```

In [4]:
```
from IPython.display import display_html
def display_side_by_side(*args):
    html_str=''
    for df in args:
        html_str+=df.to_html()
    display_html(html_str.replace('table','table style="display:inline"'),raw=
True)
```

In [5]:
```
levendata = {}
def levenshtein(s, t):
    if s == "":
        return len(t)
    if t == "":
        return len(s)
    cost = 0 if s[-1] == t[-1] else 1

    i1 = (s[:-1], t)
    if not i1 in levendata:
        levendata[i1] = levenshtein(*i1)
    i2 = (s, t[:-1])
    if not i2 in levendata:
        levendata[i2] = levenshtein(*i2)
    i3 = (s[:-1], t[:-1])
    if not i3 in levendata:
        levendata[i3] = levenshtein(*i3)
    res = min([levendata[i1]+1, levendata[i2]+1, levendata[i3]+cost])

    return res
```

In [6]:
```
def getallsimilar(query, diction, n = 6):
    a = {}
    for i in diction:
        a[i]= levenshtein(query, i)
    a = sorted(a.items(), key=operator.itemgetter(1))
    a = pd.DataFrame(a)
    return np.array(a[0].head(n))
```

In [7]:
```
def stemquery(query):
    query = query.split(" ")
    ps = PorterStemmer()
    newquery = []
    for q in query:
        newquery.append(ps.stem(q))
    return " ".join(newquery)
```

In [8]:

```
def readstemcorpus(location, typeoffile):
    with open(location, "r", encoding="utf-8") as myfile:
        stemfile=myfile.read().replace('\n', ' ')
        myfile.close()
    stemfile = stemfile.split("#")
    stemfile.remove("")
    stemfiledict = {}
    for i in stemfile:
        main = i.split(" ")
        while '' in main:
            main.remove('')
        strin = []
        if int(main[0]) < 10:
            name = "CACM-" + "0"*3 + str(main[0])
        elif int(main[0]) < 100:
            name = "CACM-" + "0"*2 + str(main[0])
        elif int(main[0]) < 1000:
            name = "CACM-" + "0"*1 + str(main[0])
        else:
            name = "CACM-" + str(main[0])

        for j in range(1,len(main)):
            if (main[j] == "pm" or main[j] == "am"):
                break
            strin.append(main[j])
        stemfiledict[name] = " ".join(strin)
    unigramdict = defaultdict(list)
    documentandlen = {}
    for i,j in stemfiledict.items():
        tokens = word_tokenize(j)
        token=[token.lower() for token in tokens if (re.match(r'^[a-zA-Z0-9][
 A-Za-z0-9-]*$', token))]
        documentandlen[i] = len(token)
        countwordsdoc = 0
        for j in range(len(token)):
            countwordsdoc +=1
            string = token[j]
            if string in unigramdict:
                tempi = str(i)
                if tempi in  unigramdict[string][0]:
                    unigramdict[string][0][tempi] += 1
                else:
                    unigramdict[string][0][tempi] = 1

            else:
                tempi = str(i)
                b = {}
                unigramdict[string].append(b)
                unigramdict[string][0][tempi] = 1


        documentandlen[i] = countwordsdoc
    return unigramdict, documentandlen
```

In [9]:

```
def psuedorel(query, location, typeoffile, stopwords = []):
    query = query.lower()
    psuedodict , psuedolen = fetchunigramdict(location, typeoffile, filelist =
[], stopwords = stopwords)
    an1 = BM25(query, documentandlen = psuedolen, uniindex = psuedodict)
    an11 = sorted(an1.items(), key=operator.itemgetter(1), reverse = True)
    an11 = pd.DataFrame(an11)
    an11 = an11.head(100)
    x = np.array(an11.head(100)[0])
    for i in range(len(x)):
        x[i] = x[i] + ".html"

    for i in range(10):
        psuedodict , psuedolen = fetchunigramdict(location, typeoffile, typ =
1, filelist = np.array(x), stopwords = stopwords)
        an1 = BM25(query, documentandlen = psuedolen, uniindex = psuedodict)
        an11 = sorted(an1.items(), key=operator.itemgetter(1), reverse = True)
        an11 = pd.DataFrame(an11)
        an11 = an11.head(100)
        x = np.array(an11.head(100)[0])
        for i in range(len(x)):
            x[i] = x[i] + ".html"
        termcount = {}
        for i,j in psuedodict.items():
                sum1 = 0
                for j,k in psuedodict[i][0].items():
                    sum1 += k
                termcount[i] = sum1
        termc = sorted(termcount.items(), key=operator.itemgetter(1), reverse
= True)
        termc = pd.DataFrame(termc)
        for j in range(len(termc)):
            kc = 0
            qst = query.split(" ")
            for q in qst:
                if (q == termc[0][j]):
                    kc = 1
                    break
            if kc == 1:
                continue
            if kc == 0:
                query = query + " " + termc[0][j]
                break
    return query
```

In [10]:

```
def tfidf(query, documentandlen, uniindex):
    a = {}
    for i,j in documentandlen.items():
        a[i] = 0
    query = tokenizequery(query)
    q = {}
    for i in query:
        i = i.lower()
        if i in q:
            q[i] += 1
        else:
             q[i] = 1

    for i in query:
        if i not in uniindex:
            continue
        for k,l in documentandlen.items():
            if k not in uniindex[i][0]:
                continue
            tf = uniindex[i][0][k] / documentandlen[k]
            idf = np.log(len(documentandlen) / len(uniindex[i][0]))
            a[k] += tf*idf
    return a
```

In [11]:

```
def querylikeDirichlet(query, documentandlen, uniindex):
    a = {}
    totaldoc = 0

    for i,j in documentandlen.items():
        a[i] = 1
        totaldoc += j

    query = tokenizequery(query)
    q = {}
    for i in query:
        i = i.lower()
        if i in q:
            q[i] += 1
        else:
             q[i] = 1
    termcount = {}

    for i,j in uniindex.items():
        sum1 = 0
        for j,k in uniindex[i][0].items():
            sum1 += k
        termcount[i] = sum1

    for i in query:
        if i not in uniindex:
            continue
        for k,l in documentandlen.items():
            mu = sum(documentandlen.values()) / len(documentandlen)

            if k in uniindex[i][0]:
                top = uniindex[i][0][k] + mu * (termcount[i]/sum(documentandle
n.values()))
                down = documentandlen[k] + mu
                a[k] += np.log(top/down)
            else:
                top = 0 + mu * (termcount[i]/sum(documentandlen.values()))
                down = documentandlen[k] + mu
                a[k] += np.log(top/down)

    return a
```

In [12]:

```
def BM25(query, documentandlen, uniindex, k1 = 1.2, k2 = 100, b = 0.75, R = 0
):
    a = {}
    for i,j in documentandlen.items():
        a[i] = 0

    query = tokenizequery(query)
    q = {}

    N = len(documentandlen)

    avdl = 0
    for i,j in documentandlen.items():
        avdl += j
    avdl /= len(documentandlen)

    for i in query:
        i = i.lower()
        if i in q:
            q[i] += 1
        else:
             q[i] = 1

    for i,j in q.items():
        if i not in uniindex:
            continue
        ni = len(uniindex[i][0])
        ri = 0
        qfi = j

        for k,l in documentandlen.items():
            if k in uniindex[i][0]:
                fi = uniindex[i][0][k]
            else:
                fi = 0


            K = k1 * ((1-b) + b * (l/avdl))

            a1 = 0.5 / (R - ri + 0.5)
            a2 = (ni - ri + 0.5) / (N - ni - R + ri + 0.5)
            a3 = (k1 + 1) * fi
            a4 = K + fi
            a5 = (k2 + 1) * qfi
            a6 = k2 + qfi
            ans = math.log(a1/a2)
            ans = ans*a3*a5/(a4*a6)

            a[k] += ans



    return a
```

In [13]:
```
def makefiledict(location, typeoffile, typ = 2, filelist = [], stopwords =
[]):
    diction = {}
    if (typ == 2):
        filelist = os.listdir(location)
    for i in filelist:
        if i.endswith(typeoffile):
            with open(location + i, "r", encoding="utf-8") as myfile:
                contents=myfile.read().replace('\n', ' ')
                myfile.close()
            contents = re.sub(pattern, "", contents)
            tokens = word_tokenize(contents)
            token=[token.lower() for token in tokens if (re.match(r'^[a-zA-Z0-
9][ A-Za-z0-9-]*$', token))]
            diction[i.split(typeoffile)[0]] = ""
            newstring = []
            for j in range(len(token)):

                string = token[j]
                if string in stopwords:
                    continue
                if (string == "html" or string == "pre"):
                    continue
                if (string == "pm" or string == "am"):
                    break
                newstring.append(string)

            diction[i.split(typeoffile)[0]] = " ".join(newstring)
    return diction
```

In [14]:
```
def fetchunigramdict(location, typeoffile, typ = 2, filelist = [], stopwords =
[]):

    unigramdict = defaultdict(list)
    documentandlen = {}
    if (typ == 2):
        filelist = os.listdir(location)
    for i in filelist:
        if i.endswith(typeoffile):
            with open(location + i, "r", encoding="utf-8") as myfile:
                contents=myfile.read().replace('\n', ' ')
                myfile.close()
            contents = re.sub(pattern, "", contents)
            tokens = word_tokenize(contents)
            token=[token.lower() for token in tokens if (re.match(r'^[a-zA-Z0-
9][ A-Za-z0-9-]*$', token))]
            documentandlen[i.split(typeoffile)[0]] = len(token)
            countwordsdoc = 0
            for j in range(len(token)):

                string = token[j]
                if string in stopwords:
                    continue
                if (string == "html" or string == "pre"):
                    continue
                countwordsdoc += 1
                if (string == "pm" or string == "am"):
                    break


                if string in unigramdict:
                    tempi = str(i.split(typeoffile)[0])
                    if tempi in  unigramdict[string][0]:
                        unigramdict[string][0][tempi] += 1
                    else:
                        unigramdict[string][0][tempi] = 1

                else:
                    tempi = str(i.split(typeoffile)[0])
                    b = {}
                    unigramdict[string].append(b)
                    unigramdict[string][0][tempi] = 1


            documentandlen[i.split(typeoffile)[0]] = countwordsdoc
    return unigramdict, documentandlen
```

In [15]:
```
def queryfetcher(location):
    unigramdict = defaultdict(list)
    queries = []
    a = ""
    coudoc = 0

    with open(location, "r", encoding="utf-8") as myfile:
        contents=myfile.read().replace('\n', ' ')
        myfile.close()
    contents = re.sub(pattern, "", contents)
    tokens = word_tokenize(contents)
    token=[token.lower() for token in tokens if (re.match(r'^[a-zA-Z0-9][ A-Za
-z0-9-]*$', token))]

    for j in range(len(token)):

        string = token[j]

        if (string == "doc"):
            continue

        if (string != "docno" and coudoc == 1):
            continue
        if (string == "docno"):
            coudoc += 1
        else:
            if (a!=""):
                a += " " + string
            else:
                a += string

        if (coudoc == 2):
            if (a != ""):
                queries.append(a)
            coudoc = 0
            a = ""
    queries.append(a)
    return queries
```

In [16]:
```
def getstopwords(location):
    with open(location, "r", encoding="utf-8") as myfile:
        contents=myfile.read().replace('\n', ' ')
        myfile.close()
    stopwords = contents.split(" ")
    stopwords = list(filter(None, stopwords))
    for i in range(len(stopwords)):
        stopwords[i] = stopwords[i].lower()
    return stopwords
```

In [17]:
```
def stopquery(query, stopwords = []):
    query = query.split(" ")
    newquery = []
    for i in range(len(query)):
        query[i] = query[i].lower()
        if query[i] not in stopwords:
            newquery.append(query[i])
    return " ".join(newquery)
```

In [18]:
```
def readstemquery(location):
    with open(location) as f:
        stemquery = f.readlines()
    stemquery = [x.strip() for x in stemquery]
    stemquery = [x.lower() for x in stemquery]
    return stemquery
```

In [19]:
```
def getsnippet(query, diction, filename, k = 10):
    comparestring = diction[filename]
    comparestring = comparestring.split(" ")
    answerstring = ""
    if k >= len(comparestring):
        query = tokenizequery(query)
        newstring = []
        for j in comparestring:
            if j in query:
                    newstring.append(color.GO + j + color.END)
            else:
                newstring.append(j)
        answerstring = " ".join(newstring)
    else:
        query = tokenizequery(query)
        maxcount = 0
        for i in range(0, len(comparestring)-k+1):
            maxc = 0
            newstring = []
            for j in range(i, i+k):
                if comparestring[j] in query:
                    maxc += 1
                    newstring.append(color.GO + comparestring[j] + color.END)
                else:
                    newstring.append(comparestring[j])
            if (maxc > maxcount):
                maxcount = maxc
                answerstring = " ".join(newstring)
    return answerstring
```

In [20]:
```
def tokenizequery(query):
    tokens = word_tokenize(query)
    token=[token.lower() for token in tokens if (re.match(r'^[a-zA-Z0-9][ A-Za
-z0-9-]*$', token))]
    return token
```

In [21]:
```
def fetchrel(location):
    with open(location) as f:
        content = f.readlines()
    content = [x.strip() for x in content]
    reldict = defaultdict(list)
    for i in content:
        i = i.split(" ")
        reldict[i[0]].append(i[2])
    return reldict
```

In [22]:
```
def createstemdictionary(listwords):
    stemdiction = defaultdict(list)
    ps = PorterStemmer()
    for i in listwords:
        if ps.stem(i) not in stemdiction:
            stemdiction[ps.stem(i)].append(i)
        else:
            stemdiction[ps.stem(i)].append(i)
    return stemdiction
```

In [23]:
```
def getstemqueries(query, stemdiction, n = 10):
    query = tokenizequery(query)
    ps = PorterStemmer()
    c = 0
    for i in query:
        if ps.stem(i) in stemdiction:
            for j in stemdiction[ps.stem(i)]:
                if j not in query:
                    query.append(j)
                    c += 1
                    break

        if (c == n):
            break
    return " ".join(query)
```

PHASE 1

TASK - 1

In [24]:
```
stopwordslist = getstopwords(location = "corpus/common_words.txt")
```

In [25]:
```
cacmq = queryfetcher("cacmquery/cacm.query.txt")
```

In [26]:
```
cacmunigramdict, cacmlen = fetchunigramdict(location = "cacm/", typeoffile =
".html")
```

In [27]:
```
reldict = fetchrel("corpus/cacm.rel.txt")
```

BM25

In [81]:
```
evalue = pd.DataFrame(columns=['Query', 'Avg Precision', 'Rank Reciprocal', 'P
recision at 5', 'Precision at 20'])
precrecdict = {}
for i in range(len(cacmq)):
    an1 = BM25(cacmq[i], documentandlen = cacmlen, uniindex = cacmunigramdict)
    an11 = sorted(an1.items(), key=operator.itemgetter(1), reverse = True)
    an11 = pd.DataFrame(an11)
    s = "Query_" + str(i+1)
    f= open("Phase 1/Task_1/BM25/Query Results/" + s + ".txt","w+",encoding="u
tf-8")
    f.write("QUERY = \"" + cacmq[i] + "\"\n")
    for j in range(100):
        k = i+1
        f.write(str((str(k) + " Q0 " + str(an11[0][j]) +" "+ str(j+1) +" " + s
tr(an11[1][j]) + " BM25" + "\n")))
    f.close()
    if str(i+1) not in reldict:
        continue
    evalue.loc[-1] = [cacmq[i],
                      getavgprecision(an11, reldict, i+1) ,
                      reciprocalrank(an11, reldict, i+1),
                      precatk(an11, reldict, i+1, 5),
                      precatk(an11, reldict, i+1, 20)]
    evalue.index = evalue.index + 1
    precrecdict[str(i+1)] = get_full_precision_recall_table(an11, reldict, i+1
)
    precrecdict[str(i+1)].to_csv(r'Phase 1/Task_1/BM25/Precision and Recall/Qu
ery_' + str(i+1) + '.csv',index=False)
evalue = evalue.reset_index(drop = True)
f= open("Phase 1/Task_1/BM25/MAP.txt","w+")
f.write("MAP = ")
f.write(str(evalue['Avg Precision'].mean()))
f.close()
f= open("Phase 1/Task_1/BM25/MRR.txt","w+")
f.write("MRR = ")
f.write(str(evalue['Rank Reciprocal'].mean()))
f.close()
evalue[['Query','Precision at 5', 'Precision at 20']].to_csv(r'Phase 1/Task_1/
BM25/P@K.csv',index=False)
```

TF-IDF

In [112]:
```
evalue = pd.DataFrame(columns=['Query', 'Avg Precision', 'Rank Reciprocal', 'P
recision at 5', 'Precision at 20'])
precrecdict = {}
for i in range(len(cacmq)):
    an1 = tfidf(cacmq[i], documentandlen = cacmlen, uniindex = cacmunigramdict
)
    an11 = sorted(an1.items(), key=operator.itemgetter(1), reverse = True)
    an11 = pd.DataFrame(an11)
    s = "Query_" + str(i+1)
    f= open("Phase 1/Task_1/TF-IDF/Query Results/" + s + ".txt","w+",encoding=
"utf-8")
    f.write("QUERY = \"" + cacmq[i] + "\"\n")
    for j in range(100):
        k = i+1
        f.write(str((str(k) + " Q0 " + str(an11[0][j]) +" "+ str(j+1) +" " + s
tr(an11[1][j]) + " tf-idf" + "\n")))
    f.close()
    if str(i+1) not in reldict:
        continue
    evalue.loc[-1] = [cacmq[i],
                      getavgprecision(an11, reldict, i+1) ,
                      reciprocalrank(an11, reldict, i+1),
                      precatk(an11, reldict, i+1, 5),
                      precatk(an11, reldict, i+1, 20)]
    evalue.index = evalue.index + 1
    precrecdict[str(i+1)] = get_full_precision_recall_table(an11, reldict, i+1
)
    precrecdict[str(i+1)].to_csv(r'Phase 1/Task_1/TF-IDF/Precision and Recall/
Query_' + str(i+1) + '.csv',index=False)
evalue = evalue.reset_index(drop = True)
f= open("Phase 1/Task_1/TF-IDF/MAP.txt","w+")
f.write("MAP = ")
f.write(str(evalue['Avg Precision'].mean()))
f.close()
f= open("Phase 1/Task_1/TF-IDF/MRR.txt","w+")
f.write("MRR = ")
f.write(str(evalue['Rank Reciprocal'].mean()))
f.close()
evalue[['Query','Precision at 5', 'Precision at 20']].to_csv(r'Phase 1/Task_1/
TF-IDF/P@K.csv',index=False)
```

Query Likelihood Model [Dirichlet Smoothing]

In [115]:
```
evalue = pd.DataFrame(columns=['Query', 'Avg Precision', 'Rank Reciprocal', 'P
recision at 5', 'Precision at 20'])
precrecdict = {}
for i in range(len(cacmq)):
    an1 = querylikeDirichlet(cacmq[i], documentandlen = cacmlen, uniindex = ca
cmunigramdict)
    an11 = sorted(an1.items(), key=operator.itemgetter(1), reverse = True)
    an11 = pd.DataFrame(an11)
    s = "Query_" + str(i+1)
    f= open("Phase 1/Task_1/QLMDirichlet/Query Results/" + s + ".txt","w+",enc
oding="utf-8")
    f.write("QUERY = \"" + cacmq[i] + "\"\n")
    for j in range(100):
        k = i+1
        f.write(str((str(k) + " Q0 " + str(an11[0][j]) +" "+ str(j+1) +" " + s
tr(an11[1][j]) + " QueryLikelihoodDirichlet" + "\n")))
    f.close()
    if str(i+1) not in reldict:
        continue
    evalue.loc[-1] = [cacmq[i],
                      getavgprecision(an11, reldict, i+1) ,
                      reciprocalrank(an11, reldict, i+1),
                      precatk(an11, reldict, i+1, 5),
                      precatk(an11, reldict, i+1, 20)]
    evalue.index = evalue.index + 1
    precrecdict[str(i+1)] = get_full_precision_recall_table(an11, reldict, i+1
)
    precrecdict[str(i+1)].to_csv(r'Phase 1/Task_1/QLMDirichlet/Precision and R
ecall/Query_' + str(i+1) + '.csv',index=False)
evalue = evalue.reset_index(drop = True)
f= open("Phase 1/Task_1/QLMDirichlet/MAP.txt","w+")
f.write("MAP = ")
f.write(str(evalue['Avg Precision'].mean()))
f.close()
f= open("Phase 1/Task_1/QLMDirichlet/MRR.txt","w+")
f.write("MRR = ")
f.write(str(evalue['Rank Reciprocal'].mean()))
f.close()
evalue[['Query','Precision at 5', 'Precision at 20']].to_csv(r'Phase 1/Task_1/
QLMDirichlet/P@K.csv',index=False)
```

TASK - 2

Query Time Stemming

In [80]:
```
z['computer']
```

Out[80]:
```
['computerization', 'computerized', 'computerize']
```

In [79]:
```
z = createstemdictionary(cacmunigramdict.keys())
```

In [174]:
```
evalue = pd.DataFrame(columns=['Query', 'Avg Precision', 'Rank Reciprocal', 'P
recision at 5', 'Precision at 20'])
precrecdict = {}
stemqueries = []
for i in range(len(cacmq)):
    stemqueries.append(getstemqueries(cacmq[i], z))
    an1 = BM25(stemqueries[i], documentandlen = cacmlen, uniindex = cacmunigra
mdict)
    an11 = sorted(an1.items(), key=operator.itemgetter(1), reverse = True)
    an11 = pd.DataFrame(an11)
    s = "Query_" + str(i+1)
    f= open("Phase 1/Task_2/QueryStemming/Query Results/" + s + ".txt","w+",en
coding="utf-8")
    f.write("QUERY = \"" + stemqueries[i] + "\"\n")
    for j in range(100):
        k = i+1
        f.write(str((str(k) + " Q0 " + str(an11[0][j]) +" "+ str(j+1) +" " + s
tr(an11[1][j]) + " BM25" + "\n")))
    f.close()
    if str(i+1) not in reldict:
        continue
    evalue.loc[-1] = [stemqueries[i],
                      getavgprecision(an11, reldict, i+1) ,
                      reciprocalrank(an11, reldict, i+1),
                      precatk(an11, reldict, i+1, 5),
                      precatk(an11, reldict, i+1, 20)]
    evalue.index = evalue.index + 1
    precrecdict[str(i+1)] = get_full_precision_recall_table(an11, reldict, i+1
)
    precrecdict[str(i+1)].to_csv(r'Phase 1/Task_2/QueryStemming/Precision and
 Recall/Query_' + str(i+1) + '.csv',index=False)
evalue = evalue.reset_index(drop = True)
f= open("Phase 1/Task_2/QueryStemming/MAP.txt","w+")
f.write("MAP = ")
f.write(str(evalue['Avg Precision'].mean()))
f.close()
f= open("Phase 1/Task_2/QueryStemming/MRR.txt","w+")
f.write("MRR = ")
f.write(str(evalue['Rank Reciprocal'].mean()))
f.close()
evalue[['Query','Precision at 5', 'Precision at 20']].to_csv(r'Phase 1/Task_2/
QueryStemming/P@K.csv',index=False)
```

Psuedo Relevance Feedback

In [119]:
```
evalue = pd.DataFrame(columns=['Query', 'Avg Precision', 'Rank Reciprocal', 'P
recision at 5', 'Precision at 20'])
precrecdict = {}
newquery = []
for i in range(len(cacmq)):
    newquery.append(psuedorel(query = cacmq[i], location = "cacm/", typeoffile
=".html", stopwords = stopwordslist))
    an1 = BM25(newquery[i], documentandlen = cacmlen, uniindex = cacmunigramdi
ct)
    an11 = sorted(an1.items(), key=operator.itemgetter(1), reverse = True)
    an11 = pd.DataFrame(an11)
    s = "Query_" + str(i+1)
    f= open("Phase 1/Task_2/PsuedoRelevance/Query Results/" + s + ".txt","w+",
encoding="utf-8")
    f.write("QUERY = \"" + newquery[i] + "\"\n")
    for j in range(100):
        k = i+1
        f.write(str((str(k) + " Q0 " + str(an11[0][j]) +" "+ str(j+1) +" " + s
tr(an11[1][j]) + " BM25" + "\n")))
    f.close()
    if str(i+1) not in reldict:
        continue
    evalue.loc[-1] = [newquery[i],
                      getavgprecision(an11, reldict, i+1) ,
                      reciprocalrank(an11, reldict, i+1),
                      precatk(an11, reldict, i+1, 5),
                      precatk(an11, reldict, i+1, 20)]
    evalue.index = evalue.index + 1
    precrecdict[str(i+1)] = get_full_precision_recall_table(an11, reldict, i+1
)
    precrecdict[str(i+1)].to_csv(r'Phase 1/Task_2/PsuedoRelevance/Precision an
d Recall/Query_' + str(i+1) + '.csv',index=False)
evalue = evalue.reset_index(drop = True)
f= open("Phase 1/Task_2/PsuedoRelevance/MAP.txt","w+")
f.write("MAP = ")
f.write(str(evalue['Avg Precision'].mean()))
f.close()
f= open("Phase 1/Task_2/PsuedoRelevance/MRR.txt","w+")
f.write("MRR = ")
f.write(str(evalue['Rank Reciprocal'].mean()))
f.close()
evalue[['Query','Precision at 5', 'Precision at 20']].to_csv(r'Phase 1/Task_2/
PsuedoRelevance/P@K.csv',index=False)
```

TASK-3

STOPPING

In [126]:
```
stopwordslist = getstopwords(location = "corpus/common_words.txt")
cacmq = queryfetcher("cacmquery/cacm.query.txt")
stopdict, stoplen = fetchunigramdict(location = "cacm/", typeoffile = ".html",
stopwords = stopwordslist)
```

Stopping - BM25

In [127]:
```
evalue = pd.DataFrame(columns=['Query', 'Avg Precision', 'Rank Reciprocal', 'P
recision at 5', 'Precision at 20'])
precrecdict = {}
stoppedqueries = []
for i in range(len(cacmq)):
    stoppedqueries.append(stopquery(cacmq[i], stopwords = stopwordslist))
    an1 = BM25(stoppedqueries[i], documentandlen = stoplen, uniindex = stopdic
t)
    an11 = sorted(an1.items(), key=operator.itemgetter(1), reverse = True)
    an11 = pd.DataFrame(an11)
    s = "Query_" + str(i+1)
    f= open("Phase 1/Task_3/Stopping/BM25/Query Results/" + s + ".txt","w+",en
coding="utf-8")
    f.write("QUERY = \"" + stoppedqueries[i] + "\"\n")
    for j in range(100):
        k = i+1
        f.write(str((str(k) + " Q0 " + str(an11[0][j]) +" "+ str(j+1) +" " + s
tr(an11[1][j]) + " BM25" + "\n")))
    f.close()
    if str(i+1) not in reldict:
        continue
    evalue.loc[-1] = [stoppedqueries[i],
                      getavgprecision(an11, reldict, i+1) ,
                      reciprocalrank(an11, reldict, i+1),
                      precatk(an11, reldict, i+1, 5),
                      precatk(an11, reldict, i+1, 20)]
    evalue.index = evalue.index + 1
    precrecdict[str(i+1)] = get_full_precision_recall_table(an11, reldict, i+1
)
    precrecdict[str(i+1)].to_csv(r'Phase 1/Task_3/Stopping/BM25/Precision and
 Recall/Query_' + str(i+1) + '.csv',index=False)
evalue = evalue.reset_index(drop = True)
f= open("Phase 1/Task_3/Stopping/BM25/MAP.txt","w+")
f.write("MAP = ")
f.write(str(evalue['Avg Precision'].mean()))
f.close()
f= open("Phase 1/Task_3/Stopping/BM25/MRR.txt","w+")
f.write("MRR = ")
f.write(str(evalue['Rank Reciprocal'].mean()))
f.close()
evalue[['Query','Precision at 5', 'Precision at 20']].to_csv(r'Phase 1/Task_3/
Stopping/BM25/P@K.csv',index=False)
```

TF-IDF

In [128]:
```
evalue = pd.DataFrame(columns=['Query', 'Avg Precision', 'Rank Reciprocal', 'P
recision at 5', 'Precision at 20'])
precrecdict = {}
stoppedqueries = []
for i in range(len(cacmq)):
    stoppedqueries.append(stopquery(cacmq[i], stopwords = stopwordslist))
    an1 = tfidf(stoppedqueries[i], documentandlen = stoplen, uniindex = stopdi
ct)
    an11 = sorted(an1.items(), key=operator.itemgetter(1), reverse = True)
    an11 = pd.DataFrame(an11)
    s = "Query_" + str(i+1)
    f= open("Phase 1/Task_3/Stopping/TF-IDF/Query Results/" + s + ".txt","w+",
encoding="utf-8")
    f.write("QUERY = \"" + stoppedqueries[i] + "\"\n")
    for j in range(100):
        k = i+1
        f.write(str((str(k) + " Q0 " + str(an11[0][j]) +" "+ str(j+1) +" " + s
tr(an11[1][j]) + " tf-idf" + "\n")))
    f.close()
    if str(i+1) not in reldict:
        continue
    evalue.loc[-1] = [stoppedqueries[i],
                      getavgprecision(an11, reldict, i+1) ,
                      reciprocalrank(an11, reldict, i+1),
                      precatk(an11, reldict, i+1, 5),
                      precatk(an11, reldict, i+1, 20)]
    evalue.index = evalue.index + 1
    precrecdict[str(i+1)] = get_full_precision_recall_table(an11, reldict, i+1
)
    precrecdict[str(i+1)].to_csv(r'Phase 1/Task_3/Stopping/TF-IDF/Precision an
d Recall/Query_' + str(i+1) + '.csv',index=False)
evalue = evalue.reset_index(drop = True)
f= open("Phase 1/Task_3/Stopping/TF-IDF/MAP.txt","w+")
f.write("MAP = ")
f.write(str(evalue['Avg Precision'].mean()))
f.close()
f= open("Phase 1/Task_3/Stopping/TF-IDF/MRR.txt","w+")
f.write("MRR = ")
f.write(str(evalue['Rank Reciprocal'].mean()))
f.close()
evalue[['Query','Precision at 5', 'Precision at 20']].to_csv(r'Phase 1/Task_3/
Stopping/TF-IDF/P@K.csv',index=False)
```

STEMMED CORPUS

In [129]:
```
stemdict, stemlen = readstemcorpus(location = "corpus/cacm_stem.txt", typeoffi
le = ".txt")
stemquery = readstemquery(location = "corpus/cacm_stem.query.txt")
```

BM25

In [130]:
```
evalue = pd.DataFrame(columns=['Query', 'Avg Precision', 'Rank Reciprocal', 'P
recision at 5', 'Precision at 20'])
precrecdict = {}
for i in range(len(stemquery)):
    an1 = BM25(stemquery[i], documentandlen = stemlen, uniindex = stemdict)
    an11 = sorted(an1.items(), key=operator.itemgetter(1), reverse = True)
    an11 = pd.DataFrame(an11)
    s = "Query_" + str(i+1)
    f= open("Phase 1/Task_3/Stemmed/BM25/Query Results/" + s + ".txt","w+",enc
oding="utf-8")
    f.write("QUERY = \"" + stemquery[i] + "\"\n")
    for j in range(100):
        k = i+1
        f.write(str((str(k) + " Q0 " + str(an11[0][j]) +" "+ str(j+1) +" " + s
tr(an11[1][j]) + " BM25" + "\n")))
    f.close()
    if str(i+1) not in reldict:
        continue
    evalue.loc[-1] = [stemquery[i],
                      getavgprecision(an11, reldict, i+1) ,
                      reciprocalrank(an11, reldict, i+1),
                      precatk(an11, reldict, i+1, 5),
                      precatk(an11, reldict, i+1, 20)]
    evalue.index = evalue.index + 1
    precrecdict[str(i+1)] = get_full_precision_recall_table(an11, reldict, i+1
)
    precrecdict[str(i+1)].to_csv(r'Phase 1/Task_3/Stemmed/BM25/Precision and R
ecall/Query_' + str(i+1) + '.csv',index=False)
evalue = evalue.reset_index(drop = True)
f= open("Phase 1/Task_3/Stemmed/BM25/MAP.txt","w+")
f.write("MAP = ")
f.write(str(evalue['Avg Precision'].mean()))
f.close()
f= open("Phase 1/Task_3/Stemmed/BM25/MRR.txt","w+")
f.write("MRR = ")
f.write(str(evalue['Rank Reciprocal'].mean()))
f.close()
evalue[['Query','Precision at 5', 'Precision at 20']].to_csv(r'Phase 1/Task_3/
Stemmed/BM25/P@K.csv',index=False)
```

TF-IDF

In [131]:
```
evalue = pd.DataFrame(columns=['Query', 'Avg Precision', 'Rank Reciprocal', 'P
recision at 5', 'Precision at 20'])
precrecdict = {}
for i in range(len(stemquery)):
    an1 = tfidf(stemquery[i], documentandlen = stemlen, uniindex = stemdict)
    an11 = sorted(an1.items(), key=operator.itemgetter(1), reverse = True)
    an11 = pd.DataFrame(an11)
    s = "Query_" + str(i+1)
    f= open("Phase 1/Task_3/Stemmed/TF-IDF/Query Results/" + s + ".txt","w+",e
ncoding="utf-8")
    f.write("QUERY = \"" + stemquery[i] + "\"\n")
    for j in range(100):
        k = i+1
        f.write(str((str(k) + " Q0 " + str(an11[0][j]) +" "+ str(j+1) +" " + s
tr(an11[1][j]) + " tf-idf" + "\n")))
    f.close()
    if str(i+1) not in reldict:
        continue
    evalue.loc[-1] = [stemquery[i],
                      getavgprecision(an11, reldict, i+1) ,
                      reciprocalrank(an11, reldict, i+1),
                      precatk(an11, reldict, i+1, 5),
                      precatk(an11, reldict, i+1, 20)]
    evalue.index = evalue.index + 1
    precrecdict[str(i+1)] = get_full_precision_recall_table(an11, reldict, i+1
)
    precrecdict[str(i+1)].to_csv(r'Phase 1/Task_3/Stemmed/TF-IDF/Precision and
Recall/Query_' + str(i+1) + '.csv',index=False)
evalue = evalue.reset_index(drop = True)
f= open("Phase 1/Task_3/Stemmed/TF-IDF/MAP.txt","w+")
f.write("MAP = ")
f.write(str(evalue['Avg Precision'].mean()))
f.close()
f= open("Phase 1/Task_3/Stemmed/TF-IDF/MRR.txt","w+")
f.write("MRR = ")
f.write(str(evalue['Rank Reciprocal'].mean()))
f.close()
evalue[['Query','Precision at 5', 'Precision at 20']].to_csv(r'Phase 1/Task_3/
Stemmed/TF-IDF/P@K.csv',index=False)
```

PHASE - 2 [Snippet Generation and Highlighting]

In [38]:
```
diction2 = makefiledict(location = "cacm/", typeoffile = ".html")
```

In [86]:
```
p2query = input("Enter Your Query : ")
phase2 = BM25(p2query, documentandlen = cacmlen, uniindex = cacmunigramdict)
phase21 = sorted(phase2.items(), key=operator.itemgetter(1), reverse = True)
phase21 = pd.DataFrame(phase21)
for i in range(10):
    print("\n")
    print("Result:",i+1)
    print("Document Name :", color.BOLD + phase21[0][i] + color.END)
    print(getsnippet(p2query, diction2, phase21[0][i], k = 10))
```

Enter Your Query : algorithms rule

Result: 1
Document Name : **CACM-2273**
tables by rule mask method without rule mask two algorithms

Result: 2
Document Name : **CACM-2239**
numerical integration integration rule adaptive integration automatic integra
tion simpsons rule

Result: 3
Document Name : **CACM-2074**
numerical integration integration rule adaptive integration automatic integra
tion simpsons rule

Result: 4
Document Name : **CACM-2263**
optimal and near-optimal flowcharts two new algorithms two new algorithms

Result: 5
Document Name : **CACM-2009**
simpsons rule for multiple integration algorithm 233 d1 cacm august

Result: 6
Document Name : **CACM-1672**
the integration of periodic analytic functions by the trapezoidal rule

Result: 7
Document Name : **CACM-1909**
numerical integration of its integral representation using the trapezoidal ru
le

Result: 8
Document Name : **CACM-0570**
simpsons rule integrator algorithm 103 cacm june 1962 kuncir g

Result: 9
Document Name : **CACM-1058**
simpsons rule for multiple integration algorithm 233 cacm june 1964

Result: 10
Document Name : **CACM-0429**
adaptive nimerical integration by simpsons rule algorithm 145 cacm december

PHASE - 3

STOPPING WITH EXPANSION

In [71]:
```
stopwordslist = getstopwords(location = "corpus/common_words.txt")
cacmq = queryfetcher("cacmquery/cacm.query.txt")
stopdict, stoplen = fetchunigramdict(location = "cacm/", typeoffile = ".html",
stopwords = stopwordslist)
```

In [124]:
```
evalue = pd.DataFrame(columns=['Query', 'Avg Precision', 'Rank Reciprocal', 'P
recision at 5', 'Precision at 20'])
precrecdict = {}
newquery = []
stoppedqueries = []
for i in range(len(cacmq)):
    stoppedqueries.append(stopquery(cacmq[i], stopwords = stopwordslist))
    newquery.append(psuedorel(query = stoppedqueries[i], location = "cacm/", t
ypeoffile=".html", stopwords = stopwordslist))
    an1 = BM25(newquery[i], documentandlen = stoplen, uniindex = stopdict)
    an11 = sorted(an1.items(), key=operator.itemgetter(1), reverse = True)
    an11 = pd.DataFrame(an11)
    s = "Query_" + str(i+1)
    f= open("Phase 3/StoppingExpansion/Query Results/" + s + ".txt","w+",encod
ing="utf-8")
    f.write("QUERY = \"" + newquery[i] + "\"\n")
    for j in range(100):
        k = i+1
        f.write(str((str(k) + " Q0 " + str(an11[0][j]) +" "+ str(j+1) +" " + s
tr(an11[1][j]) + " BM25" + "\n")))
    f.close()
    if str(i+1) not in reldict:
        continue
    evalue.loc[-1] = [newquery[i],
                      getavgprecision(an11, reldict, i+1) ,
                      reciprocalrank(an11, reldict, i+1),
                      precatk(an11, reldict, i+1, 5),
                      precatk(an11, reldict, i+1, 20)]
    evalue.index = evalue.index + 1
    precrecdict[str(i+1)] = get_full_precision_recall_table(an11, reldict, i+1
)
    precrecdict[str(i+1)].to_csv(r'Phase 3/StoppingExpansion/Precision and Rec
all/Query_' + str(i+1) + '.csv',index=False)
evalue = evalue.reset_index(drop = True)
f= open("Phase 3/StoppingExpansion/MAP.txt","w+")
f.write("MAP = ")
f.write(str(evalue['Avg Precision'].mean()))
f.close()
f= open("Phase 3/StoppingExpansion/MRR.txt","w+")
f.write("MRR = ")
f.write(str(evalue['Rank Reciprocal'].mean()))
f.close()
evalue[['Query','Precision at 5', 'Precision at 20']].to_csv(r'Phase 3/Stoppin
gExpansion/P@K.csv',index=False)
```

EVALUATION

In [71]:
```
def reciprocalrank(results, reldict, queryid):
    for i in range(len(results)):
        if results[0][i] in reldict[str(queryid)]:
            return 1/(i+1)
    return 0
```

In [72]:
```
def getavgprecision(results, reldict, queryid):
    rel = 0
    ans = 0
    for i in range(len(results)):
        if results[0][i] in reldict[str(queryid)]:
            rel +=1
            ans += rel/(i+1)
    if (rel == 0):
        return 0
    return ans/rel
```

In [73]:
```
def precatk(results, reldict, queryid, k):
    rel = 0
    for i in range(k):
        if results[0][i] in reldict[str(queryid)]:
            rel +=1
    return rel/k
```

In [74]:
```
def get_full_precision_recall_table(results, reldict, queryid):
    evalue = pd.DataFrame(columns=["Result","Precision", "Recall"])
    rel = 0
    reldoc = len(reldict[str(queryid)])
    if (reldoc == 0):
        print(str(queryid))
    for i in range(100):
        if results[0][i] in reldict[str(queryid)]:
            rel +=1
        evalue.loc[-1] = [results[0][i],rel/(i+1),rel/reldoc]
        evalue.index = evalue.index + 1
    return evalue.reset_index(drop = True)
```

EXTRA-CREDIT [Spelling Correction]

In [87]:
```
diction1 = makefiledict(location = "cacm/", typeoffile = ".html")
```

In [89]:

```
querycheck = 0
while (querycheck == 0):
    quer = input("Enter your Query : ")
    quer1 = tokenizequery(quer)
    diction = cacmunigramdict.keys()
    n = 6
    for j in range(n):
        counter = 0
        queryz = []
        k = 0
        for i in quer1:
            if i not in cacmunigramdict.keys():
                counter = 1
                simwords = getallsimilar(i, diction)
                queryz.append(color.BOLD + simwords[j] + color.END)
                k = 1
            else:
                queryz.append(i)
        if (k == 1):
            print("Did you Mean : "," ".join(queryz))
            print('\n')
    if (counter == 0):
        querycheck = 1
    if (querycheck == 1):
        ar = BM25(" ".join(queryz), documentandlen = cacmlen, uniindex = cacmu
nigramdict)
        ar = sorted(ar.items(), key=operator.itemgetter(1), reverse = True)
        ar = pd.DataFrame(ar)
        for i in range(10):
            print("\n")
            print("Result:",i+1)
            print("Document Name :", color.BOLD + ar[0][i] + color.END)
            print(getsnippet(" ".join(queryz), diction1, ar[0][i], k = 10))
```

Enter your Query : algorthm computre is good tiem
Did you Mean : **algorithm compute** is good **them**

Did you Mean : **algorithms computers** is good **ties**

Did you Mean : **algorithmic computer** is good **tied**

Did you Mean : **althm compare** is good **the**

Did you Mean : **algol computed** is good **ibm**

Did you Mean : **aegerter computes** is good **time**

Enter your Query : algorithm compute is good them

Result: 1
Document Name : **CACM-2460**
clenshaw-curtis quadrature algorithm r424 cacm august 1973 good a j

Result: 2
Document Name : **CACM-2986**
grammar can be done by a practical algorithm that is

Result: 3
Document Name : **CACM-1525**
a power of two computing time for this algorithm is

Result: 4
Document Name : **CACM-3069**
analysis algorithm a new interprocedural data flow analysis algorithm is

Result: 5
Document Name : **CACM-2520**
multipliers algorithm c386 cacm april 1973 ragland l c good

Result: 6
Document Name : **CACM-2547**
pictorial features contour maps region coverage and line structures is

Result: 7
Document Name : **CACM-0222**
coding of external symbols into symbols internal to a compute

Result: 8
Document Name : **CACM-1206**
sjp is used to discard wrong information and to compute

Result: 9
Document Name : **CACM-1909**
numerical integration it is shown to be practical to compute

Result: 10
Document Name : **CACM-1619**
discussed computational error generated by some algorithms used to compute