A Project Description

On

# Boolean Retrieval System


**ABHIRAJ KHARE** **2020A7PS0161H**

**RAHUL JAUHARI** **2020A7PS0106H**

**SANCHIT GUPTA** **2020A7PS2069H**


Under the supervision of

**FACULTY: Dr. N L Bhanumurthy**

**SUBMITTED AS AN EVALUATION COMPONENT OF**

**Course: Information Retrieval CS F469**


**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI**

**HYDERABAD CAMPUS**

## Tokenizing and Removing Punctuation:

The Input format of the below code is in the form of 'String'.

The Output format of the below code is in the form of the 'List'.

```
# Removes all the punctutation marks/special characters from the text
text = self.punctuationMarks(text)

# Tokenize text into words
words = word_tokenize(text)
```

The above code tokenizes the given string, removes punctuation and outputs a 'List' which does not have punctuations.

## Removing Stop words:

The input of the below code is in the form of 'List'.

Output is also in the form of a 'List'.

Here we are using the output of the above 'Tokenizing' function and passed the output as parameter to 'stopwords_clr' function.

```
# Remove stopwords
# convert remaining words to lowercase
words = [word.lower() for word in words if word not in self.stopword]
```

The above function removes all the stop words and outputs a 'List' which don't contain stop words.

## Stemming the document:

Here the input is given in the form 'List'.

Output is given in the form of 'List'.

Here we used the output of ‘stopwords_clr’ function above and passed it as a parameter to ‘stem_tokens’

```
#Stemming
for word in words:
    self.reverse_stem[self.ps.stem(word)].append(word)
for key in self.reverse_stem.keys():
    self.reverse_stem[key] = self.unique(self.reverse_stem[key])
words = [self.ps.stem(word) for word in words]
```

The output of the above function is a ‘List’ of words which are stemmed.

## Inverted Index:

Here the input is in the form of a set of documents (List) and index of the document (Int).

The output we get is a dictionary which have key-value pairs, where words of the documents are keys and posting list of the keys are the values of the dictionary respectively.

```
        # Add posting to Final Posting List
        for term in terms:
            self.postings[term].append(i)

        # Make a list of indexed documents
        self.doc[i] = os.path.basename(filename)

        i = i + 1

    # Making inverted index out of final posting list.
    self.dictionary = self.postings.keys()
```

In the above code we passed a folder containing 4 documents. All the documents are first tokenized, stemmed and the stop words are removed. We got a dictionary containing key-value pairs of words of document and their respective posting lists.

## Spelling Correction:

Here the below function, ‘nearest_word’ takes the input as a String for which we used Levenshtein distance to find the nearest word in the document if any.

```
""" Performing a spell check and correction on all the query
words if the spelling is wrong. This is done by comparing
the edit distance between the query words with all the
unique words across all the documents. The word is then
replaced by the word which has the minimum edit distance and
among those, the one having the largest posting list.
If the query word exists in the documents, the minimum edit
distance is zero and the word remains unchanged. """
count=0
for key in self.dictionary:
    distance= minEditDistance(key,token,len(key),len(token))
    if distance <= threshold:
        count=count+1
        for term in self.reverse_stem[key]:
            if(threshold >= minEditDistance(term,token,len(term),len(token))):
                keys.append(term)
if count == 0:
    print( token," is not found in the corpus!" )
    return np.zeros(len(self.doc), dtype=bool)

word.append(self.bits(searched_token,token,keys))
```

```
# Correcting user queries to recieve the right answers using  minimum edit distance algorithm (Levenshte
#Levenshtein Distance is found in bottom right corner of each matrix
# 1 for each insertion, 1 for deletion and 1 for substitution
"""
 Distance -> Minimum number of edits (operations) required to convert 'str1' into 'str2':
 1. Insert -> Cost =1
 2. Remove -> Cost =1
 3. Replace -> Cost =1
"""
def minEditDistance(a, b, rows, cols):

    #Build empty matrix of correct size to store results
    distance = arr = [[0 for i in range(cols+1)] for j in range(rows+1)]

    for i in range(rows + 1):
        for j in range(cols + 1):

            # First string is empty, insert all characters of second string
            if i == 0:
                distance[i][j] = j

            # Second string is empty, remove all characters of second string
            elif j == 0:
                distance[i][j] = i

            # If last characters are same, ignore last char and recur for remaining string
            elif a[i-1] == b[j-1]:
                distance[i][j] = distance[i-1][j-1]

            # If last character are different, consider all possibilities and find minimum
            else:
                distance[i][j] = min( 1+distance[i-1][j],  # Remove
                                      1+distance[i][j-1],  # Insert
                                      1+distance[i-1][j-1])  # Replace

    return distance[rows][cols]
```

Output of the above function is a String which is the closest word in the document.

## Querying:

## 1.Wildcard Query:

Here the below function, ‘wildcard_process’ takes the input in the form of String.

It gives Output of list of the documents in which the wildcard is present.

```
def matchwildcard(str1):
    obj=BooleanIRSystem("./corpus/*")
    inverted_index=obj.postings

    str1=str1.lower()
    str1="$"+str1+"$"
    query=list(ngrams(str1, 2))
    '''bigrams of query terms are claculated using ngram method of nltk'''
    for bigram in query:
        temp=[]
        if('*' in bigram):
            if((bigram[0]=='*') and (bigram[1]=='*')):
                continue
            elif(bigram[0]=='*'):
                for word in bigramindex.keys():
                    for bigramlist in bigramindex[word]:
                        if(bigram[1]==bigramlist[1]):
                            temp.append(word)
            elif(bigram[1]=='*'):
                for word in bigramindex.keys():
                    for bigramlist in bigramindex[word]:
                        if(bigram[0]==bigramlist[0]):
                            temp.append(word)
        else:
            for word in bigramindex.keys():
                if(bigram in bigramindex[word]):
                    temp.append(word)
        inverted_index=AND(inverted_index, temp)
    '''the bigrams of query term are matched with that of all unique words in dataset'''

    print('matching words:')
    print(inverted_index)
    '''matching words are printed'''
    model = BooleanIRSystem("./corpus/*")
    for word in inverted_index:
        print(model.query(word))
    return set(inverted_index)
```


## 2.Boolean Query:

Here the below function, ‘boolean_query’ takes the input in the form of a Query (String) and the Inverted Index (Dictionary).

In [8]:
```
from Model import BooleanIRSystem
import time

if __name__ == "__main__":
    model = BooleanIRSystem("./corpus/*")
    start_time = time.time()
    """ Accepting query as input from the user and splitting
    the query into boolean words (&,|,~) and
    query words """
    print(model.query(input("Search Query:")))
    end_time = time.time()
    total_time = end_time - start_time
    print("Entering Query + Searching Time: ",total_time)
```

```
Preprocessing + Indexing Time:  29.037702798843384
Search Query:(richard & henry) | romeo
Searching Time:  2.70437240600586
['henry-iv-part-1_TXT_FolgerShakespeare.txt', 'henry-iv-part-2_TXT_FolgerShakespeare.txt', 'henry-vi-part-1_TXT_FolgerShakespea
re.txt', 'henry-vi-part-2_TXT_FolgerShakespeare.txt', 'henry-vi-part-3_TXT_FolgerShakespeare.txt', 'henry-viii_TXT_FolgerShakes
peare.txt', 'henry-v_TXT_FolgerShakespeare.txt', 'king-john_TXT_FolgerShakespeare.txt', 'richard-iii_TXT_FolgerShakespeare.tx
t', 'richard-ii_TXT_FolgerShakespeare.txt', 'romeo-and-juliet_TXT_FolgerShakespeare.txt', 'the-taming-of-the-shrew_TXT_FolgerSh
akespeare.txt']
Entering Query + Searching Time:  20.63770055770874
```

In [9]:
```
from Model import BooleanIRSystem
import time

if __name__ == "__main__":
    model = BooleanIRSystem("./corpus/*")
    start_time = time.time()
    """ Accepting query as input from the user and splitting
    the query into boolean words (&,|,~) and
    query words """
    print(model.query(input("Search Query:")))
    end_time = time.time()
    total_time = end_time - start_time
    print("Entering Query + Searching Time: ",total_time)
```

```
Preprocessing + Indexing Time:  29.323376893997192
Search Query:juliet | ~shakepeare
shakepear  is not found in the corpus!
Did you mean these ? :
shakespeare
Giving results based on:  shakespeare
Searching Time:  2.12913131713867
['measure-for-measure_TXT_FolgerShakespeare.txt', 'romeo-and-juliet_TXT_FolgerShakespeare.txt']
Entering Query + Searching Time:  15.994519710540771
```

The above function gives output of ‘List’ of documents which fulfil the query.