<u>REPORT</u>
<u>ASSIGNMENT-2</u>

- **<u>Data Preprocessing Steps:</u>**

- Stop Word Removal
- Lemmatization
- Removal of words of length less than 3.
- Removing empty lines
- Removing multiple spaces between words
- Removing Special Characters
- Lower case conversion

- **<u>Methodology For Question-1:</u>**

1. The Jaccard Coefficient for a document is calculated as follows,

   JC = X intersection Y/ X union Y

   - Here, X is our document and Y is our query, X intersection Y indicates the count of how many words in the given query "Y" are matching with document "X".
   - The Problem i encountered with JC based scoring is that when the size or length of the document is small the JC of the document tends to be higher due to small value in the denominator.
   - Documents retrieved based on jaccard score for a particular query are attached in the analysis part.
   -

2. The TF-IDF score for the document is calculated by various variants. I used the following 5 variants to compute it. If the term is not present in the corpus the tf will be 0. So tf-idf=0
   - **Raw count variate of both Tf and Df**
     Tf = frequency of the term in that document , if not present assigned 0.
     Df = log(total number of docs/frequency of the term in the doc)
   - **Log normalization variate of tf and inverse variation of df**
     Tf = log(1+Tf), if not present assigned 0.
     Df = log(total number of docs/frequency of the term in the doc)
   - **Binary weighting scheme for tf and inverse variation of df**
     Tf = if term is present in the doc tf=1 other wise tf=00.
     Df = log(total number of docs/frequency of the term in the doc)
   - **Normalized term frequency (dividing by total term frequency count in that doc after pre-proc)for tf and inverse variation of df¶**
     Tf = frequency of the term in that document / sum of total tfs in that doc.
     Df = log(total number of docs/frequency of the term in the doc)
   - **Double normalization(0.5) for tf and inverse variation of df**
     Tf = 0.5+(0.5* tf that term in that document / max tf in that doc)
     Df = log(total number of docs / frequency of the term in the doc)

<u>All the results for the above variants are calculated with giving weightage to titles and without giving weightage to the titles.</u>

- Here we find the tf-idf for each term in the query and sum all of them w.r.t to a document and assign that scalar as a tf-idf score for that document.
- Now we order the documents based on descending order of tf-idf scores.

- **<u>Justification and Assumptions:</u>**

  1. I have found that **double normalization with 0.5** has been working reasonably well when compared to other variants of tf-idf, i can justify this by considering 5 random queries with their ground truths being the name of their respective files from which they have been directly taken.
  2. Most of the times the docs retrieved using double normalization were present at the top most position if not at least among the top-3 relevant docs. Which is happening with other variants.
  3. If the term in the query is present in any title's document that **doc is given 20% more weightage compared to other documents.**

3. 1**.**The Cosine-Similarity score for the documents is assigned based on the double normalization variant of tf and inverse variation of DF.here i calculated the tf-idf vector for the query and also for each document. Both the vectors are of query vector size only.
Dot product = A.B/|A|.|B|
2. Now we perform dot product between query vector and the document vector and assign that dot product value as a score to that document.
3. In this way we assign scores to all the docs and retrieve the docs based on descending order of similarity scores.
4. **<u>Calculated Cosine Similarity Scores by giving weightage to titles and without giving weightage to the titles.</u>**

- **<u>Methodology for Question-2</u>**

1. Converted the query into lower case as the entire English U.K dictionary is in complete lower case.
2. Insertion cost = 2 ; Deletion Cost = 1 , Substitution = 3
3. Used dynamic programming paradigm for table construction with below formulation,

   D[i,j] = min(D[i-1,j-1] + replace_cost , D[i-1,j] + del_cost, D[i,j-1] + ins_cost)

5. Results have been attached in analysis.