# Report

Assignment - 2

---

Mona Singh - MT21053
Nehal Chourasia - MT21056

## Question 1 - [40 Points] Scoring and Term-Weighting

### Assumptions:

There are no duplicate files present in the dataset.
The document indexing starts from 0.

### PreProcessing Steps:

Normalization : All the text is converted into lower case format.

Tokenization : The text is split into smaller units using nltk library.

StopWord Removal : All the stop words from the English language list from the nltk library are removed from the list.

Punctuation removal: All the special symbols are removed including the numbers 0-9.

Whitespace removal : All the white spaces are removed if any are present.

Lemmatization : It stems from the word but makes sure that the word does not lose its meaning.

Preprocessing returns a list of all the clean words.

## PART A : Jaccard Coefficient [20 points]

### Methodology:

- For every document, create a token list that contains the preprocessed terms in that particular document. Create a dataframe with the document name, preprocessed terms and the length as columns.
- Jaccard coefficient is calculated by the following formulae Jaccard Coefficient = Intersection of (doc,query) / Union of (doc,query). Therefore preprocess the query and get the token list for the query.
- Find the ***intersection*** and the ***union*** of the query with each document term list.

- Apply the formula for jaccard and store it.
- Sort from largest to smallest according to the jaccard values and return the first 5.

```
def relevance(query):
    import joblib
    # print("Jaccard_coeff")
    query_token_list = preProcess(query)

    doc_df_preprocessed = joblib.load('doc_df_preprocessed.joblib')
                                      (variable) doc_df_preprocessed: Any
    doc_df_preprocessed['union'] = doc_df_preprocessed.apply(lambda row:list(set(row['doc_token']) | set(query_token_list)),axis=1 )
    doc_df_preprocessed['intersection'] = doc_df_preprocessed.apply(lambda row:list(set(row['doc_token']) & set(query_token_list)),axis=1 )
    doc_df_preprocessed['jaccard_coeff'] = doc_df_preprocessed.apply(lambda row:len(row['intersection'])/len(row['union']),axis=1 )
    # doc_df_preprocessed.set_index('doc_name')
    return doc_df_preprocessed



""" Find the relevance with the query using the dataframe"""
doc_df_preprocessed = relevance('jokes rules attends received')

""" Top 5 """
top5 = doc_df_preprocessed.nlargest(5, ['jaccard_coeff'])
print('top5 relevant docs--->\n',top5[['jaccard_coeff']])
```

```
top5 relevant docs--->
              jaccard_coeff
doc_name
languag.jok        0.015873
temphell.jok       0.015152
calif.hum          0.014706
gaiahuma           0.012821
hedgehog.txt       0.012579
```

## PART B : TF-IDF Matrix [20 points]

## Methodology:

- Find the unique words in the dataset and store it as corpus words.
- Create a dataframe with corpus words as index and document name as the columns.
- Iterate over each (word,document) pair in the dataframe and count the frequency of that word in that particular document with the help of document token list.
- For different weighting scheme, fill the matrix differently\
    - Binary: if the word is in the document's token list fill (word,document) =1 else 0.
    - Raw count: fill (word,document) with the count of words in the document's token list.
    - Term Frequency: fill (word,document) with count(word in the document)/ total words in document.
    - Log normalization : fill (word,document) with log(1+frequency of term in the document)
    - Double normalization : fill (word,document) with 0.5+0.5*(term frequency in document d / max(term frequency in document d)
- Find the IDF values by dividing the total number of documents by the number of documents containing a term t.

- Multiply the idf & tf values and then sort the documents according to the values. Display top 5.

```
scores,results =TF_IDF('polio genovese chest')
scores
```

| | Top1 | Top2 | Top3 | Top4 | Top5 |
|---|---|---|---|---|---|
| Binary | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 |
| Raw count | 6.000000 | 6.000000 | 5.000000 | 4.000000 | 4.000000 |
| Term frequency | 0.006969 | 0.004926 | 0.004630 | 0.004053 | 0.003911 |
| Log normalization | 1.945910 | 1.945910 | 1.791759 | 1.609438 | 1.609438 |
| Double normalization | 0.515544 | 0.510363 | 0.507576 | 0.505848 | 0.505181 |

```
results
```

| | Top1 | Top2 | Top3 | Top4 | Top5 |
|---|---|---|---|---|---|
| Binary | gameshow.txt | facedeth.txt | insults1.txt | cucumber.jok | barney.txt |
| Raw count | practica.txt | quest.hum | facedeth.txt | luzerzo2.hum | prac3.jok |
| Term frequency | ghostfun.hum | age.txt | back1.txt | luzerzo2.hum | t_zone.jok |
| Log normalization | practica.txt | quest.hum | facedeth.txt | luzerzo2.hum | prac3.jok |
| Double normalization | quest.hum | prac3.jok | t_zone.jok | practica.txt | back1.txt |

Jaccard Coefficient has a major drawback when the two sets being compared have different sizes.
TF-IDF is computationally cheap.
TF-IDF cannot help carry semantic meaning
pros and cons of using each scoring scheme
Binary : It captures only the presence or absence of the word
Raw Count: It captures the frequency of the occurrence of words
Term Frequency : It captures the relation of the word in the document  with the other words present in the same document
Log Normalization :  Compared to the the large range of values in raw count, log normalization narrows the range
Double Normalization : This technique captures the relation between each term frequency and the maximum frequency term in the documents.

## Question 2 - [25 points] Ranked-Information Retrieval and Evaluation

## Methodology:

- Filter the dataset such that it contains only the rows for qid:4

- Make a dataframe using the filtered dataset with column relevance score.
- Sort the dataframe by the relevance score in the descending order and save it.
- Count the frequency of relevance 0,1,2 and 3.
- To find the total number of files, multiply the fact(count_0)*fact(count_1)*fact(count_2)*fact(count_3).
- To find the DCG, iterate over the dataframe and apply the formula

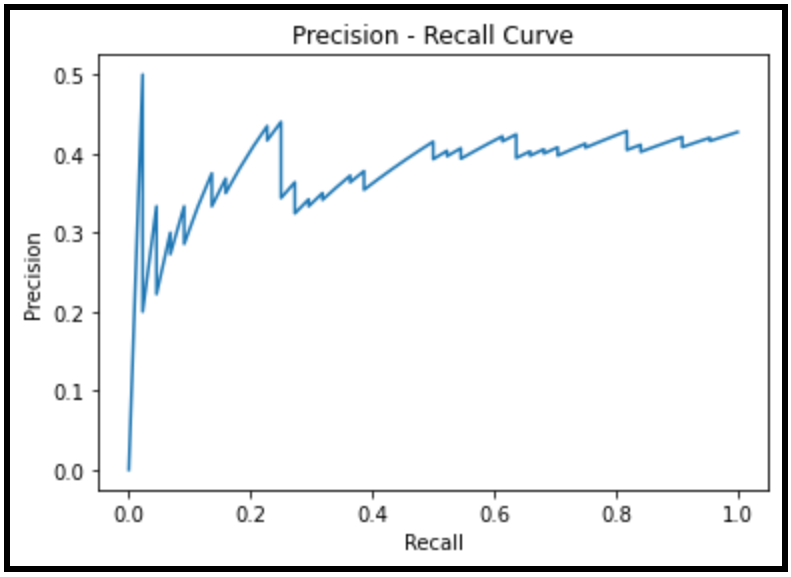
$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)} = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2(i+1)}$$

  as we move down the rows.
- Sort the dataframe in descending order of the relevance score and apply the same formula to get IDCG.
- Get nDCG = DCG/IDCG

```
"""MaxDCG"""
maxDCG = DCG(final_data,len(final_data))

"""nDCG for 50 values"""
trueDCG50 = DCG(final_data,51)
idealDCG50 = DCG(ideal_data,51)

"""nDCG for whole dataset"""
trueDCG = DCG(final_data,len(final_data))
idealDCG = DCG(ideal_data,len(ideal_data))

nDCG50 = trueDCG50/idealDCG50
nDCG = trueDCG/idealDCG

print("Max DCG is: {}".format(maxDCG))
print("nDCG at 50: {}".format(nDCG50))
print("nDCG whole Dataset: {}".format(nDCG))
```

```
Max DCG is: 12.550247459532576
nDCG at 50: 0.41082175342157357
nDCG whole Dataset: 0.6976332021320715
```

- For the precision and recall curve, calculate the precision as the number of relevant documents/ total number of documents retrieved as we iterate through the dataframe.
- Recall is calculated by the number of relevant retrieved documents divided by the number of total relevant documents.
- Append both the values in their respective lists while iterating.
- Plot the curve.




## Question 3 - [ 35 points ] Naive Bayes Classifier

**Dataset:** 20_newsgroups.zip

Dataset is downloaded and loaded into the notebook and documents belonging only [comp.graphics, sci.med, talk.politics.misc, rec.sport.hockey, sci.space] are selected for text classifier.

**PreProcessing Steps:**

- Normalization : All the text is converted into lower case format.
- Tokenization : The text is split into smaller units using nltk library.
- StopWord Removal : All the stop words from the English language list from the nltk library are removed from the list.
- Punctuation removal: All the special symbols are removed including the numbers 0-9.
- Whitespace removal : All the white spaces are removed if any are present.
- Lemmatization : It stems from the word but makes sure that the word does not lose its meaning.

Preprocessing returns a list of all the clean words for each document.

**Test and train DataSplit:**

Split is done using train_test_split of sklearn.model_selection

- The data is split into 80:20 train -test ratio
- The data is split into 70:30 train -test ratio
- The data is split into 50:50 train -test ratio

**TF-ICF calculation:**

TF-ICF score for a given term belonging to a class can be calculated as follows:

- Term Frequency (TF): Number of occurrences of a term in all documents of a particular class
- Class Frequency (CF): Number of classes in which that term occurs
- Inverse-Class Frequency (ICF): log( N / CF), where N represents the number of classes

Based on the Tf-icf value , words with top k value are selected for further analysis.

Here , K= 10

**Top k Features  from each 5 classes are :**

{'capolitics', 'hallam', 'nsmca', 'simtel', 'shuttle', 'sciastro', 'auroraalaskaedu', 'abortion', 'sinus', 'hammerl', 'scorer', 'fbi', 'health', 'geb', 'optilinkcom', 'quicktime', 'alchemychemutorontoca', 'vertex', 'photoshop', 'playoff', 'pixel', 'altsciplanetary', 'orbit', 'recsporthockey', 'allergy', 'altpoliticsclinton', 'billboard', 'kinsey', 'altsex', 'talkreligionmisc', 'baalke', 'kilometer', 'pov', 'ramseycslaurentianca', 'lyme', 'sky', 'sabre', 'chi', 'altbinariespicturesd', 'gilmour', 'cramer', 'altconspiracy', 'kelvinjplnasagov', 'antibiotic', 'astronaut', 'venus', 'lunar', 'zootorontoedu', 'bitmap', 'batf', 'tiff', 'talkabortion', 'ahl', 'candida', 'talkpoliticsmisc', 'pittuucp', 'noring', 'compmultimedia', 'texture', 'misclegal', 'scispaceshuttle', 'oiler', 'compgraphics', 'yeast', 'hockey', 'bruin', 'lilley', 'scoring', 'socmen', 'jpeg', 'scimed', 'altactivism', 'msg', 'hiv', 'phill', 'compgraphicsanimation', 'stl', 'rind', 'topazucscedu', 'siggraph', 'maynard', 'spacecraft', 'pluto', 'altfanrushlimbaugh', 'infection', 'optilink', 'xv', 'dcx', 'gif', 'nfotis', 'higgins', 'raster', 'mccall', 'image', 'scienergy', 'canuck', 'vga', 'vitamin', 'det', 'goalie', 'nyi', 'comet', 'stephanopoulos', 'espn', 'recfoodcooking', 'hicnet', 'altsciphysicsnewtheories', 'coach', 'scianthropology', 'nhl', 'clayton', 'jfif', 'polygon', 'prb', 'pit', 'dyer', 'dscomsadesyde', 'patient', 'altpoliticslibertarian', 'cspittedu', 'scispace', 'altgraphicspixutils', 'migraine', 'puck'}

**Naive Bayes algorithm :**

Multinomial naive bayes are implemented from scratch.
The model is trained on train data and accuracy is found on the different test datas.

- **80:20 split**

**Accuracy** 0.995

**Classification matrix**

| | **precision** | **recall** | **f1-score** | **support** |
|---|---|---|---|---|
| 0 | 0.98 | 0.99 | 0.99 | 196 |
| 1 | 1.00 | 1.00 | 1.00 | 205 |
| 2 | 1.00 | 0.99 | 1.00 | 206 |
| 3 | 0.99 | 0.99 | 0.99 | 199 |
| 4 | 1.00 | 1.00 | 1.00 | 194 |
| **accuracy** | | | 0.99 | 1000 |
| **macro avg** | 0.99 | 1.00 | 0.99 | 1000 |
| **weighted avg** | 1.00 | 0.99 | 1.00 | 1000 |

**Confusion Matrix**

```
[[195   0   0   1   0]
 [  0 205   0   0   0]
 [  1   0 204   1   0]
 [  2   0   0 197   0]
 [  0   0   0   0 194]]
```

- **70:30 split**

**Accuracy** 0.994

**Classification matrix**

| | **precision** | **recall** | **f1-score** | **support** |
|---|---|---|---|---|
| 0 | 0.99 | 0.99 | 0.99 | 297 |
| 1 | 1.00 | 1.00 | 1.00 | 284 |
| 2 | 1.00 | 0.99 | 0.99 | 324 |
| 3 | 0.99 | 0.99 | 0.99 | 296 |
| 4 | 1.00 | 1.00 | 1.00 | 299 |

| | | | | |
|---|---|---|---|---|
| **accuracy** | | | 0.99 | 1500 |
| **macro avg** | 0.99 | 0.99 | 0.99 | 1500 |
| **weighted avg** | 0.99 | 0.99 | 0.99 | 1500 |

**Confusion Matrix**

```
[[294   0   1   2   0]
 [  0 284   0   0   0]
 [  2   0 321   1   0]
 [  2   0   0 294   0]
 [  0   1   0   0 298]]
```

- **50:50 split**

**Accuracy** 0.9948

**Classification matrix**

| **precision** | **recall** | **f1-score** | **support** |
|---|---|---|---|
| 0 | 0.99 | 0.99 | 487 |
| 1 | 1.00 | 1.00 | 511 |
| 2 | 0.99 | 1.00 | 507 |
| 3 | 0.99 | 0.99 | 495 |
| 4 | 1.00 | 1.00 | 500 |
| **accuracy** | | 0.99 | 2500 |
| **macro avg** | 0.99 | 0.99 | 2500 |
| **weighted avg** | 0.99 | 0.99 | 2500 |

**Confusion Matrix**

```
[[481   0   5   1   0]
 [  0 511   0   0   0]
 [  1   0 505   1   0]
```

```
[  3   0   0 492   0]
[  0   1   0   1 498]]
```

**Analysis:**

There is no significant change in the accuracy of the three splits.

Also from the confusion matrix , it can be seen that the mostly all values are on diagonal, that represents that the classification is 99% correct for all the three cases.