Information Retrieval Assignment 2(Group 53)

Name: Ankur Gupta
(MT21113, ankur21113@iiitd.ac.in)
Name: Nandha Shivam Niteshkumar
(MT21128, shivam21128@iiitd.ac.in)

Question 1:
1. Use the same data given in assignment 1 and carry out the same preprocessing steps as mentioned
Before.
Approach:
- First, we will import all the needed libraries for the preprocessing.
- After that, we are reading the files from the dataset
- The steps for the preprocessing are as follows.

1. **Remove punctuation:** Here RegEx package removes the punctuations like ‘’, ‘ /n /t from all.
2. **Tokenizer:** Using the nltk inbuilt libraries, we are converting the entire file content into tokens.
3. **Lower case:** Using the inbuilt function lower(), we convert all text into lower text.
4. **Remove Stop words:** We remove the stop words from all files using the stopwords library of nltk package.
5. **Lemmatization:** Using the nltk package, we perform the lemmatization on all the files.
6. **Remove blank space tokens:** With the strip function's help, we remove the space from the dataset.

- Now simply calling all the functions on the query and storing the result in s.

2. To calculate this, make a set of the document token and query token and perform intersection and union between the query and each document.

Approach:

- Now to perform the intersection between the document token and the query token we are anding both terms together.
- To find the union between them, we add both term lengths and subtract the intersection length.
- These call things have been implemented in the function called jackardCoeff(set1,set2).
- In the user input function, we are taking a query from the user and finding jacquard coefficients between docs.

3. Report the top 5 relevant documents based on the value of the Jaccard coefficient.

Approach:

- Now from the user we are taking the input query in UserInput() function and based on the Jaccard coefficient we are ranking the 5 most relevant documents from the datasets.
- The output for the same can be seen in the following snapshot.

```
Enter the query: coffee
<class 'list'>
['coffee']
The Top 5 Documents retrieved according to Jackard Coefficient
1   recipe.012   0.015625
2   banana03.brd   0.008771929824561403
3   cooking.jok   0.008064516129032258
4   btscke04.des   0.007751937984496124
5   recipe.005   0.007575757575757576
```

TF-IDF Matrix [20 points]

1. Use the same data given in assignment 1 and carry out the same preprocessing steps as mentioned
Before.

- As can be seen in the previous part of the question we have performed all the preprocessing steps like punctuation, tokenizer, lowercase, removing stop words, blank space removal, etc.

2. Build the matrix of size no. of document x vocab size.

- Here first in the corpus set, we are storing all tokens that have been generated from the pre-processing.
- Now we are creating a dictionary and the structure of the dictionary will look like the following.
- {Token : {‘File Name’ : Frequency } }
- Now creating the empty matrix of the size of no. of document X vocab size as can be seen in the below snapshot.

	themduring	user	ehuuuuuuh	resting	nylon	leasing	teal	dubrin	queen	missouri	...	dugacvsrochesteredu	tak	fourteenth	kairos	pane
1st_aid.txt	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0
a-team	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0
a_fish_c.apo	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0
a_tv_t-p.com	0.0	2.392297	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	7.033506	0.0	0.0	0.0	0.0
abbott.txt	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
zen.txt	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0
zgtoilet.txt	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0
zodiac.hum	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0
zucantom.sal	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0
zuccmush.sal	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0

- 1133 rows × 79450 columns
- The size of the generated matrix is **1133 rows × 79450 columns**

3. Fill the tf idf values in the matrix of each word of the vocab.

- Here to find the values of tf IDF for each token in dict_token we are calculating the IDF score for each token using the below formula.

$$\mathrm{IDF(word)=log(total\ no.\ of\ documents/document\ frequency(word)+1)}$$

- Now the IDF values corresponding to the token can be seen in the below snapshot.

```
{'herbalherb1st': 7.033506484287697, 'aidcalendulacomfreyremediessickmedicin
e': 7.033506484287697, 'herbal': 4.479083539813148, 'first': 1.08870833355116
47, 'aid': 2.54005846986308, 'kit': 3.4029053240073113, 'calendula': 7.033506
484287697, 'ointment': 5.6498541326306455, 'use': 1.2299154170790678, 'mino
r': 3.0726933146901194, 'cut': 1.678533342764278, 'graz': 5.936656310612987,
'red': 1.7372712839439852, 'rash': 5.246146494219127, 'skin': 2.5120353171762
53, 'comfrey': 5.936656310612987, 'suitable': 3.5077001018987586, 'bruise':
4.96021880818224, 'damage': 2.7188252742701686, 'external': 3.87553189684573
1, 'blood': 2.207489175948102, 'vessel': 3.7248324715226877, 'vein': 3.690602
1074253825, 'st': 2.630088659632498, 'johnswort': 7.033506484287697, 'oil':
2.1061516920074097, 'beneficial': 4.6443908991413725, 'itchy': 5.092873392333
228, 'irritable': 5.6498541326306455, 'psoriasis': 6.341240749355558, 'also':
1.2164729670901675, 'good': 1.1245720923639073, 'sunburn': 5.427589702252176,
'applied': 3.0221055588088808, 'night': 1.5637429531719709, 'liver': 3.507700
1018987586, 'mixture': 2.4586207875170754, 'mild': 3.5648268054439574, 'laxat
ive': 7.033506484287697, 'property': 3.0726933146901194, 'help': 1.5033912904
028754, 'digestion': 5.427589702252176, 'rich': 2.4165570719867904, 'food':
1.8202685267864085, 'take': 1.0921813983378192, 'one': 0.8478025285785858, 't
easpoon': 3.2649108744412123, '30': 1.8595820655668163, 'minute': 1.486809319
```


4. Make the query vector of size vocab

- Here in the UserInput function, we are creating a query vector of size vocab, as can be seen from the following statement.
- list1 = [0] * len(corpus)

5. Compute the TF-IDF score for the query using the TF-IDF matrix. Report the top 5 relevant
documents based on the score.

- Here first we are going to create an empty data frame called TfIdf of size the same as the corpus.
- Now we are calculating the IDF scores for each token using the following formula.

$$\mathrm{IDF(word)}{=}\log(\text{total no. of documents}/\text{document frequency(word)}{+}1)$$

-
- Finally, to find the tf_idf score we are multiplying the term Freq with IDF hence we will be calculating the tf_idf score for each.
- Below is the output of the user query.
- 5 relevant documents based on the score are being displayed.

```
Enter Querysweetened
<class 'list'>
['sweetened']
{'insect1.txt': 0.056583003631464915, 'coffee.faq': 0.03358520280123844, 'coffe
e.txt': 0.025923446556052134, 'drinks.txt': 0.024200770194681462, 'bread.rec':
0.020962180332967973}
```

•

6. Use all 5 weighting schemes for term frequency calculation and report the TF-IDF score and results for each scheme separately.

Weighting Scheme	TF Weight
Binary	0,1
Raw count	f(t,d)
Term frequency	$f(t,d)/\sum f(t',d)$
Log normalization	log(1+f(t,d))
Double normalization	0.5+0.5*(f(t,d)/ max(f(t',d))

- Above is the list of 5 weighting schemes for the tf we have implemented and calculated TF-IDF score for the same and formula used to find TF Weight are also can be seen in the above.

- All above mentioned weighting schemes result have been separately added to the folder of question 1

->State the pros and cons of using each scoring scheme to find the relevance of documents in your
report.
<u>Jaccard Coefficient:</u>
Pros:
- Jaccard considers the measure of the overlaps between any 2 sets.
- It is a statistic used for comparing the similarity and diversity of sets A and B.
- It is very easy to calculate the Jaccard coefficient.

Cons:

- Term frequency is not considered while calculating the Jaccard coefficient.
- Jaccard coefficient ignores the rare terms from the collection which can sometimes be more informative than the frequent terms.

TF-IDF:

- Pros:
- Considers the frequency of the term using the term frequency measure of tf-idf.
- It also gives importance to the rare terms.

Cons:

- Needs a lot of calculation to derive this.
- It is based on the bag of words model hence it fails to capture semantic repeated occurrences in different documents etc.

Question 2:

1. **Consider only the queries with qid:4 and the relevance judgment labels as relevance score.**

- First, we are going to import all files from the dataset and append the file names and store it in a list called relevance.
- After that, we are fetching the only queries with qid:4 and we are storing it in a data frame df.
- Now our df will look like this

	0	1	2	3	4	5	6	7	8	9	...	129	130	131	132	133	134	135	136	137	138
0	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	128:2	129:9	130:124	131:4678	132:54	133:74	134:0	135:0	136:0	NaN
1	0	qid:4	1:3	2:0	3:3	4:0	5:3	6:1	7:0	8:1	...	128:0	129:8	130:122	131:508	132:131	133:136	134:0	135:0	136:0	NaN
2	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	128:2	129:8	130:115	131:508	132:51	133:70	134:0	135:0	136:0	NaN
3	0	qid:4	1:3	2:0	3:3	4:0	5:3	6:1	7:0	8:1	...	128:82	129:17	130:122	131:508	132:83	133:107	134:0	135:10	136:13.35	NaN
4	1	qid:4	1:3	2:0	3:3	4:0	5:3	6:1	7:0	8:1	...	128:11	129:8	130:121	131:508	132:103	133:120	134:0	135:0	136:0	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
98	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	128:35	129:1	130:153	131:4872	132:9	133:55	134:0	135:0	136:0	NaN
99	1	qid:4	1:3	2:0	3:3	4:2	5:3	6:1	7:0	8:1	...	128:367	129:6	130:153	131:2383	132:18	133:99	134:0	135:16	136:11.3166666666667	NaN
100	2	qid:4	1:2	2:0	3:2	4:0	5:2	6:0.666667	7:0	8:0.666667	...	128:0	129:0	130:49182	131:26966	132:15	133:69	134:0	135:193	136:21.9355595468361	NaN
101	1	qid:4	1:2	2:0	3:2	4:0	5:2	6:0.666667	7:0	8:0.666667	...	128:0	129:1	130:42877	131:26562	132:12	133:24	134:0	135:56	136:62.9206042323688	NaN
102	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	128:1415	129:14	130:5334	131:6434	132:4	133:17	134:0	135:0	136:0	NaN

103 rows × 139 columns

- df will contain only queries with qid:4 and the relevance judgment labels as relevance score

2. **Make a file rearranging the query-url pairs in order of max DCG. State how many such files could be made.**

- To find a DCG value firstly we are going to sort the entire data frame and store it in the output.txt file.
- Now we are finding a number of ways such files can be made in the order of MAX DCG.
- Here is the answer for the same.
-

```
[ ] print(noOfWays(maxDCGdf))

    1989349737593837059982604761490532989693684017056657058820518031270485799269519348241268656543105024000000000000000000000
```

3. **Compute nDCG**
    - To compute the Normalised Discounted Cumulative Gain we are creating a function to find DCG and in that, we are using the below formula to find Discounted Cumulative Gain.

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

(a) At 50:

- Here first we are finding the DCG(rel_50) which means we will take 50 relevant docs and finally we will find Ideal DCG and diving both will give us the nDCG at 50 which can be seen below.

- Top 50

```
[ ] #First 50
    rel_50 = list(df[0].values)[0:50]
    # print(DCG(rel_50))
    print(DCG(rel_50)/IDCG(rel_50))

    0.5253808413557646
```

(b) For the whole dataset:

- Here for the entire dataset, we are doing the same things as we have done for finding the 50 nDCG.
- The output value for the nDCG for the entire dataset can be seen below.

```
[ ] print(DCG(list(df[0].values))/IDCG(list(df[0].values)))

    0.5979226516897831
```

4. **Assume a model that simply ranks URLs on the basis of the value of feature 75 (sum of TF-IDF on the whole document) i.e. the higher the value, the more relevant the URL. Assume any non-zero relevance judgment value to be relevant. Plot a Precision-Recall curve for query "qid:4".**

- Here first for fetching the value of the feature 75 we are splitting the dataset and appending the first index of it into the feature_rel list.
- Now for relevance score, all 0 scores are being treated as non-relevant and all non-zero scores will be treated like 1 as relevant.
- Now we are sorting the values depending on the feature relevance we got.
- For finding the precision and recall we are going to use the following formula to find the value for plotting the curve.

- **Precision**: fraction of retrieved docs that are relevant =

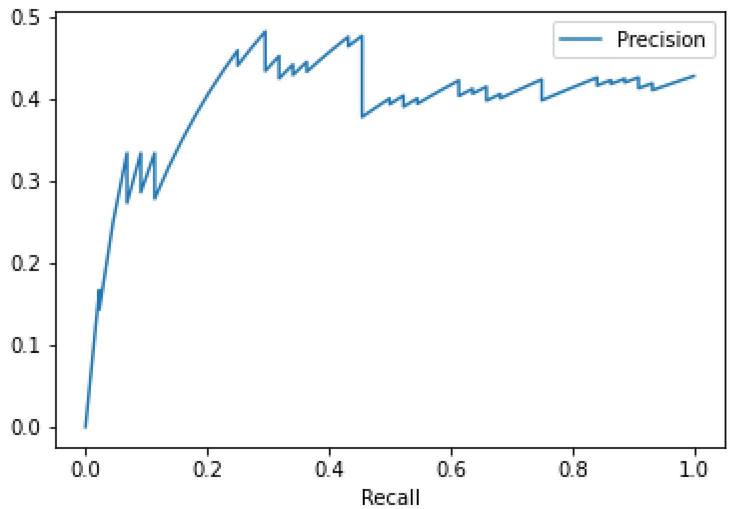
- P(relevant|retrieved) $=$ (No. of documents retrieved $\cap$ No. of relevant documents) / No. of documents retrieved

- **Recall**: fraction of relevant docs that are retrieved = P(retrieved|relevant) $=$ (No. of documents retrieved $\cap$ No. of relevant documents) / No. of relevant documents
- After calculating these all our data frames will look like this.

	Relevance	Feature Relevance	Precision	Recall
0	0	90.531710	0.000000	0.000000
1	0	538.388954	0.000000	0.000000
2	0	88.171761	0.000000	0.000000
3	0	144.564444	0.000000	0.000000
4	1	142.589323	0.000000	0.000000
...	...	...	...	...
98	0	70.460443	0.414141	0.931818
99	1	270.132330	0.410000	0.931818
100	1	296.023694	0.415842	0.954545
101	1	528.520116	0.421569	0.977273
102	0	84.625987	0.427184	1.000000

103 rows × 4 columns

- Now plotting the recall vs precision graph using the above values.



Question 3:
Approach:

- Here first from 5 classes, we are counting the number of files in it. In each class, we are having 1000 files so a total of 5000 files will be there.
- We are creating the dictionary of a list of tokens as key-value and class name corresponding to it belongs as the value of it same can be seen in the following snapshot.

	ssme	consolidated	newseaglelercnasagov	transuranic	pockete
comp.graphics	0.000000	0.000000	0.000000	0.000000	0.00000
sci.med	0.000000	0.000000	0.000000	0.000000	0.00000
talk.politics.misc	0.000000	0.000000	0.000000	0.000000	1.60943
rec.sport.hockey	0.000000	0.000000	0.000000	0.000000	0.00000
sci.space	4.828314	1.609438	3.218876	1.609438	0.00000

5 rows × 42714 columns

1. Perform suitable pre-processing steps for the given dataset.

- We have performed all the preprocessing steps like punctuation, tokenizer, lowercase, removing stop words, blank space removal, etc.

2. Split your dataset randomly into train: test ratio. You need to select the documents randomly for splitting. You are not supposed to split documents in sequential order, for instance, choosing the first 800 documents in the train set and the last 200 in the test set for the train: test ratio of 80:20.

- Here using the sklearn libraries we are randomly splitting the data into train and test sets.
- Here we have spiltited the data into an 80:20 ratio.

3. Implement the TF-ICF scoring technique for efficient feature selection. Select the top k features for each class. Subsequently, the effective vocabulary shall be the union of the top k features of each class.

- Using the formula for TF-ICF we are going to give a score to each token present in the corpus.
- TF-ICF score of the entire matrix can be seen in the following snapshot where the row represents the classes and columns represents tokens.

	ssme	consolidated	newseaglelercnasagov	transuranic	pockete
comp.graphics	0.000000	0.000000	0.000000	0.000000	0.00000
sci.med	0.000000	0.000000	0.000000	0.000000	0.00000
talk.politics.misc	0.000000	0.000000	0.000000	0.000000	1.60943
rec.sport.hockey	0.000000	0.000000	0.000000	0.000000	0.00000
sci.space	4.828314	1.609438	3.218876	1.609438	0.00000

5 rows × 42714 columns

-

- Here we are taking the first k highest-ranked features where k we have taken as 5 and storing it in a final_token list which is basically a union of the top k features of each class.

	msg	stephanopoulos	recsporthockey	shuttle	orbit	chronic
comp.graphics	0	0	0	1	0	0
sci.med	177	0	0	2	1	88
talk.politics.misc	0	261	0	0	0	0
rec.sport.hockey	4	0	527	0	0	0
sci.space	0	0	0	271	219	0

5 rows × 25 columns

4. For each class, train your Naive Bayes Classifier on the training data.

- Now for selected top features from each class, we are creating a training set.
- And we are finding the number of times tokens have occurred on that particular class.

5. Test your classifier on testing data and report the confusion matrix and overall accuracy.

- Now we are testing our classifier on testing data and below is the confusion matrix for the same.
- The below snapshot shows the accuracy of our classifier as well which is 0.991.

```
Accuracy Score 0.991
[[208   0   1   4   0]
 [  0 214   0   0   0]
 [  1   0 189   0   0]
 [  0   0   0 188   0]
 [  1   2   0   0 192]]
```

6. Perform the above steps on 50:50, 70:30, and 80:20 training and testing split ratios.

- Splitting the datasets as mentioned above.

7. Analyze the performance of the classification algorithm for the feature selection technique across different train: test ratios.

- Now repeat the same steps for the given ratios.
- Splitting at a 50:50 ratio.

```
In [147]: from sklearn.model_selection import train_test_split
          import collections
          X_train, X_test, y_train, y_test = train_test_split(df['Tokens'], df['Label'], test_size = 0.5,random_state=42)
          y=collections.Counter(y_train)
          y

Out[147]: Counter({'sci.med': 484,
                   'rec.sport.hockey': 494,
                   'sci.space': 508,
                   'comp.graphics': 522,
                   'talk.politics.misc': 492})
```

  - Here we got the accuracy of 0.966 while classifying using this ratio.

- For 70:30 ratio

```
In [148]: from sklearn.model_selection import train_test_split
          import collections
          X_train, X_test, y_train, y_test = train_test_split(df['Tokens'], df['Label'], test_size = 0.7,random_state=42)
          y=collections.Counter(y_train)
          y

Out[148]: Counter({'sci.space': 308,
                   'talk.politics.misc': 301,
                   'comp.graphics': 311,
                   'sci.med': 278,
                   'rec.sport.hockey': 302})
```

- Accuracy of 0.986 while classifying using this ratio.

- For the 80:20 ratio:

```
In [149]: from sklearn.model_selection import train_test_split
          import collections
          X_train, X_test, y_train, y_test = train_test_split(df['Tokens'], df['Label'], test_size = 0.8,random_state=42)
          y=collections.Counter(y_train)
          y

Out[149]: Counter({'sci.med': 191,
                   'comp.graphics': 204,
                   'talk.politics.misc': 213,
                   'rec.sport.hockey': 189,
                   'sci.space': 203})
```

- Accuracy of 0.99 while classifying using this ratio.
- Coming to the performance part of each classifier at different spilled ratios it turns out that when we spilled using the ratio of 80:20 we got the best accuracy of 0.99 which is 99%.