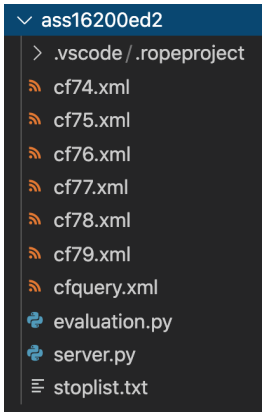
README

My answer to this assignment contains two main parts:

- The 'server.py' file which builds the web interface for query and shows the retrievd documents
- The 'evaluation.py' file which reads from 'cfquery.xml' to test my search system and evaluates it by return and print out:
  - 2 list: 'p@10' for all queries, 'PRList' average interpolated p@k for 11 recall levels
  - the MAP value

🔷How to Run

1. I didn't upload the given data, so just place them in the same folder with 'server.py' and 'evaluation.py' like this.



2. Run the 'server.py' file to enter the web interface and make a query, just run command "python server.py" in your terminal(no installation of additional modules needed), follow the link and enter your query, top 20 relevant ducuments retrieved would be presented immediately.
3. The evaluation functions are placed in the 'evaluation.py' file, run command "python evaluation.py" in your terminal and it would print out the data needed for plotting graphs. I used matplotlib and numpy module in Jupyter notebook to plot the graphs and export them as jpg(can be found in uploaded folder).

Code and graphs are as follows:

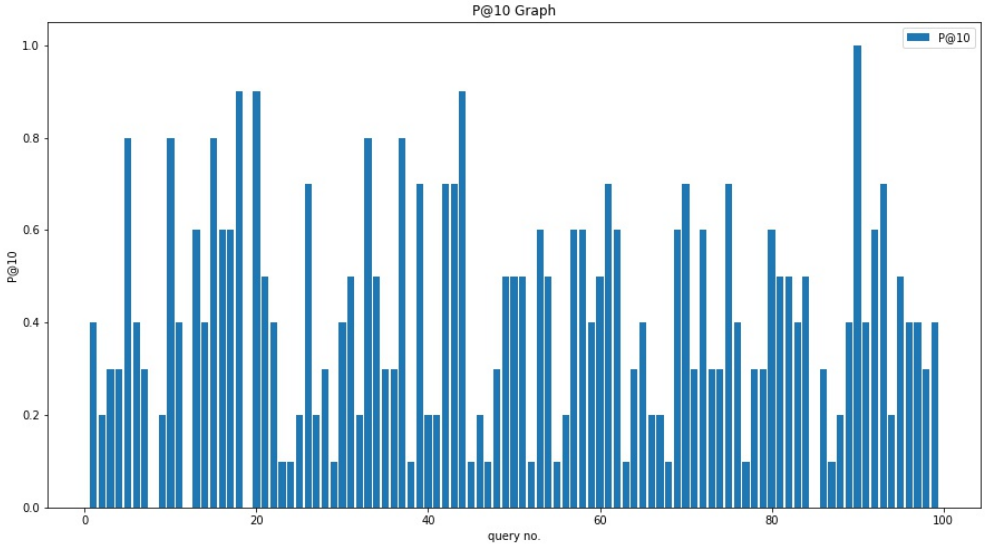
```
import matplotlib.pyplot as plt
import numpy as np

# x, y axis values points
y1 = [0.4, 0.2, 0.3, 0.3, 0.8, 0.4, 0.3, 0.0, 0.2, 0.8, 0.4, 0.0, 0.6, 0.4, 0.8, 0.6, 0.6, 0.9, 0.0, 0.9, 0.5, 0.4, 0.1, 0.1, 0.2, 0.7
x1 = [i for i in range(1, len(y1)+ 1)]
# set figure size
plt.rcParams["figure.figsize"] = (15,8)
# plotting the line 1 points
plt.bar(x1, y1, label = "P@10")

# naming the x axis
plt.xlabel('query no.')
# naming the y axis
plt.ylabel('P@10')
# giving a title to my graph
plt.title('P@10 Graph')

# show a legend on the plot
plt.legend()

# save the plot
plt.savefig('P@10.pdf')
```

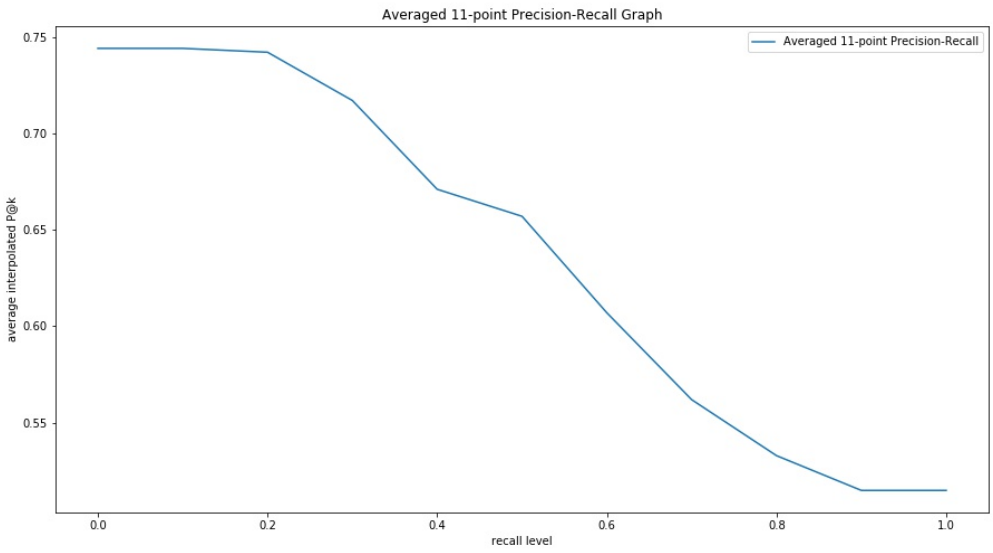



```
y1 = [0.744, 0.744, 0.742, 0.717, 0.671, 0.657, 0.607, 0.562, 0.533, 0.515, 0.515]
x1 = np.arange(0.0, 1.1, 0.1)
# set figure size
plt.rcParams["figure.figsize"] = (15,8)
# plotting the line 1 points
plt.plot(x1, y1, label = "Averaged 11-point Precision-Recall")

# naming the x axis
plt.xlabel('recall level')
# naming the y axis
plt.ylabel('average interpolated P@k')
# giving a title to my graph
plt.title('Averaged 11-point Precision-Recall Graph')

# show a legend on the plot
plt.legend()

# save the plot
plt.savefig('Avg11P_R.jpg')
```



server.py

🔹 Old functions:

The modified 'parseFile' function in last assignment that returns 4 variables, except for creating and return the inverted index for overall 6 xml files, Also return:

- total doc number(a counter that records the total document number)
- docTF dictionary: storing occurrences of every term inside one doc

  Structure: use term as key, a dictionary (use docID as key and occurrences as value) as value
- IDF dictionary storing IDF of every term from all docs

  Structure: term as key, IDF as value

🔹 New functions added:

'processQuery' Function to process query string

Logic:

1. Given query string, remove the stop words first, then remove punctuations and split into term list, remove stop words again in case some appears after handling punctuations
2. Record TF for every term in query and then norm the query TF and all terms' TF by calling 'normTF' function which divide each frequency by the total frequencies of terms
3. calculate normed TF * IDF for query terms and all document terms based on the TF, IDF dictionary returned from 'parseFile' function
4. call the 'calcCosineSimilarity' function to calculate the Cosine Similarity for every document based on this query and then return the top20 relevant docs

Finally, The flask app simply takes the query text and process it with 'processQuery' Function and then render the results on html page

evaluation.py

This file contains 2 main functions:

1. 'parseFile' Function to create inverted index from cf74-cf79 xml file
2. 'readQueryXml' Function to read every query from 'cfquery.xml' file, evaluate the predications made and return the 3 evaluation value sets

🔹 'readQueryXml' Function

Logic:

initialize the 'pAt10' and 'apAt10' list which would store the p@10 and AP@10 for each query, and 'interpolatedDct' dictionary that stores the interpolated precision for all queries, recall level as key, precision list of all queries as value

For loop to loop through every query:

extract query text to get predications list by calling 'processQuery' function, also extract the real relevant documents list

call the 'calcPrecAtK' function to get P@K and append to 'pAt10' list

call the 'calcAPAtK' function to get AP@K and append to 'apAt10' list, also the relevant document number to calculate

call the 'generateInterpolatedPList' function to calculate the average interpolated p@k for k in (0,10) and insert into 'interpolatedDct' dictionary

Finally, calculate MAP with 'apAt10' list and calculate average interpolated p@k with 'interpolatedDct' dictionary and return as 'PRList'.