Automatic Summarization of Legal Decisions using Iterative Masking of Predictive Sentences

Table of Contents

Explanation of Task -

Unsupervised Extractive Legal Document Summarization →

Given a legal case document generate a summary using unsupervised methods (here Maximal Marginal relevance)

Algorithm for the Task -

1. Obtain the IDF values using all the given documents. Now consider each document one by one.

2. Initialize summary to be empty.

3. While the word count is below the desired word count we add to summary the sentence (From a given ‘List of sentences’ = Case document in our case) with ‘maximal MMR (Maximal Marginal Relevance) score’.

4. MMR score = F(Sentence_to_be_added,Case_document)*λ - F(Sentence_to_be_added,Summary)*(1-λ).

5. Here F(Sentence_to_be_added,Some_document) =
   { i>= 0 and i < len(Some_document) }
   max(Similarity(Sentence_to_be_added,Some_document[i]))

    a. The similarity of Sentence_to_be_added is calculated with every sentence in the ‘Some_document’ and the max value is returned.

    b. For this part we used **Segment Trees**

        i. Every time we add something to the summary, we update the segment trees. (Point update)

        ii. When finding the maximum value we made a range query.

        iii. Thus for each sentence addition we reduced the complexity from O(N^2) to O(N*log(N))

Steps Followed -

1. At first we wanted to split the sentences and store it in a list. We did that by using Spacy for splitting the lines but since it was noisy we created a custom sentence splitter and splitted the sentences of a doc accordingly.

2. Next we wanted to use the tf-idf vectorizer over the list of documents, so that we can use that to obtain the tf-idf of a sentence. The tf-idf vector of a sentence will be of a particular dimension where the number of rows = 1, as there is only one sentence and the number of columns will be = the number of unique words in the whole corpus. Thus each sentence of a document can be obtained as a vector.

3. We then pre-computed the similarities between each pair of sentences in the document and stored it in a 2-D array by computing the dot product of two vectors. For each sentence the maximum similarity was thus found by comparing the values for that sentence with other sentences by using the precomputed similarity values.

4. After that we applied the Maximum Marginal Relevance(MMR) algorithm to find the summary. This algorithm was used to increase the relevancy of the sentences that are being added to the summary as well as reduce the redundancy (i.e preventing the same sentence from getting added to the summary).

5. We iterated through all the sentences in the document and found the maximum marginal relevance score of all the sentences and updated the summary by adding the sentence with the maximum marginal relevance score.

6. We kept on adding sentences to the summary unless it crossed a particular threshold which was passed as a parameter.

7. We read the files and generated the 2 summaries based on the reference summaries A1 and A2 for 3 values of λ: 0.3, 0.5, 0.7.

8. We calculate the rogue scores for all the generated summaries with respect to the reference summaries and store the rogue-1,rogue-2,rogue-l scores in ‘rogue.txt’. Rouge helps us to assess the adequacy of the summary by simply counting how many n-grams in your generated summary matches n-grams in our reference summary.

9. We calculate and display the average of rogue scores of 50 documents at the end.

Results Obtained -

Rouge average for A1 Summary :-

	Rouge-1	Rouge-2	Rouge-l
F-Score	0.5033	0.2859	0.3857
precision	0.5006	0.2843	0.5525
recall	0.5062	0.2876	0.3899

We would like to mention that in most documents we obtain ‘High’ rouge scores (>= 0.5). Only some documents had a ‘Low’ score (< 0.3), so they distorted the average very much.

Rouge average for A2 Summary :-

	Rouge-1	rouge-2	rouge-l
F-Score	0.3464	0.1709	0.2506
precision	0.3474	0.1716	0.6542
recall	0.3460	0.1704	0.2214

Interpretation of the Rouge Scores

Rouge helps us to assess the adequacy of the summary by simply counting how many n-grams in your generated summary matches n-grams in our reference summary.

Discussions-

1. Most summaries obtained had good (>=0.6) ‘rouge-l’ scores.

2. How MMR score avoids the same lines to be added to the summary multiple times.

    a. The term ‘(1-λ)*F(Sentence_to_be_added,Summary)’ prevents multiple sentences from getting added to the Summary, because if a sentence is being added to summary which is already present then F(...) = 1 and (1-λ) will be subtracted from the MMR score.

Though for some documents it was observed that the summary contained the same line multiple times. This behaviour was because these lines had high similarity with other lines, and this dominated the effect of the 2nd term.

Instructions To Execute Code -

List of Files/Folders

Sl.no	File/Folder Name	Description
1	indian-summary-len-a1.txt	Contains the name of summaries and the summary length of A1 summaries
2	indian-summary-len-a2.txt	Contains the name of summaries and the summary length of A2 summaries
3	A1(folder)	Contains A1 summary files
4	A2(folder)	Contains A2 summary files
5	sentence_splitted	Contains original text documents with splitted sentences

Instructions to execute Code

1. Put the files and folders in the same directory as the code.

2. Create 2 blank folders named **A1sum** and **A2sum** at the working directory to contain the generated summaries.

3. Change the path of the files and folders in the code accordingly-

```
X1 = pd.read_csv('./indian-summary-len-a1.txt', sep="\s", header=None)
X2 = pd.read_csv('./indian-summary-len-a2.txt', sep="\s", header=None)
with open(r'./sentence_splitted/'+X1[0][i], 'r') as file
with open(r'./'+X1[0][i], 'r') as file:
f=open(r'./A1sum/'+"sumA1_"+X1[0][i][:-4]+'_'+str(j)+".txt","w+")
f=open(r'./A2sum/'+"sumA2_"+X2[0][i][:-4]+'_'+str(j)+".txt","w+")
```

Expected Output Files

- **rogue.txt** containing the rogue scores of 50 documents

- The generated summaries will be stored in **A1sum** and **A2sum** folders

Expected Output in Terminal

- The *average rouge scores* of 50 generated summaries with A1 as reference and 50 generated summaries with A2 as reference.

Libraries Used -

1. Tfidf Vectorizer from scikit learn - To get the tf-idf vectors of each document in a collection of documents and thus computing the similarities between them. We do this by calling the tfidfvectorizer function over a corpus i.e tfidfvectorizer(corpus). In our case corpus is the list of strings where each string is a sentence.This gives us the the tf-idf vector of each document and we compute their similarity by performing suitable operations.

2. Segment_tree - We had to perform point update and range max queries, so we used segment trees to make them as fast as possible.

3. Numpy - To do the mathematical operations of the tf-idf vectors to find the similarity between them. We suppose two vectors w and v so we calculate their similarity by element wise multiplication of the matrix and compute their sum i.e np.sum(np.multiply(w,v)).

4. Rouge - To compare our results with the other reference precomputed results. After we get our summary (let it be summ) we find the similarity with given two reference summaries summA1 and summA2 by giving the command rouge.get_scores(summ,summA1) and rouge.get_scores(summ,summA2).

5. String - To remove the punctuations in a string and calculate the number of words in the sentence.

6. Sys - To increase the recursion depth limit. At first e=while calling the rouge.get_scores() function the program was giving an error that stated ‘ Maximum stack depth limit exceeded‘. We have to import the sys library and use the sys.setrecursionlimit() function and set it to a large value to avoid that error.

7. Pandas - To read the data in tabular form ,from the text file containing summary file names and the corresponding lengths.