Assumptions :

1. While removing the punctuations the letters are separated. For example u.s.a. will become u s a

2.While calculating cosine similarity between each query-document pair the document is normalized using Term frequency(tf) only while query is not normalized. In query we calculate the frequency of each word and multiply it with Inverse Document Frequency(IDF) of the word.

3.While converting number to words, if digit is greater than threshold then it is converted to words otherwise one digit is fetched from number and converted to words.

4.There are two dictionary, one for storing the words for all the documents and another for storing the words of index file. The more weightage is given to dictionary corresponds to index file (that is g is 0.6) as compared to dictionary corresponds to documents(that is g is 0.4)

5.)When user gives input 50,000 flowers as input so it comma get separated and 50,000 will split as 50 and 000 so it will be number will be converted as fifty and zero but if user gives input as 50000 flowers then it will be converted fifty thousand in words.

Tf-Idf based document retrieval:

a).Pre-Processing :

- Tokenization : Longer strings are splits into tokens. In this the text is tokenized into sentences and then sentences are tokenized into words. During tokenization the punctuations are also removed.
- After tokenizing the documents the number is converted to words(like 100 is converted to one hundred).If size is greater than each digit of number is converted to word.(like 100 is converted to one zero zero)
- Case-Folding(Lower Case) : Converting all words to lower case to make searching easy.
- Stemming : It is process of eliminating affixes from the word to get the stem word. Generally it makes word running to run.

b).Methodology :

- First step is to load the files from directory and reading the files using csopen() function.
- Once the file is read the all pre-processing is done like tokenization, conversion of number to words and stemming etc.
- For each term in document we calculate the term frequency and normalize it with length of the document and multiply it with inverse document frequency(IDF).
  TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)
  IDF(t) =1+ log(Total number of documents / Number of documents with term t in it).
- After calculating normalized tf-idf score of the term it is stored in dictionary. All the words are stored in dictionary whose keys are words and value is a also dictionary, whose key is document id and value is normalized tf-idf score of that word.
- If word is new then it is added to dictionary and if word is already exist then document is added along with normalized tf-idf of that term. Repeat till all files are read.

- Two dictionaries are maintain , one for storing words corresponds to documents while other for storing words corresponds to index file. If word is in index dictionary then it is given more weightage than word in normal dictionary.
  W (term)= g *(tf-idf(term) in index dictionary)+(1-g)*(tf-idf(term) in document dictionary)
- If user is giving query then on the basis of tf-idf score the document is sorted in descending order and top k documents are returned.

Tf-Idf based vector space document retrieval:

a). Pre-processing : Same as above.

b).Methodology :

- First step is to load the files from directory and reading the files using csopen() function.
- Once the file is read the all pre-processing is done like tokenization, conversion of number to words and stemming etc.
- For each term in document we calculate the term frequency and normalize it with magnitude of the document.
  TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)
- After calculating normalized tf score of the term it is stored in dictionary. All the words are stored in dictionary whose keys are words and value is a also dictionary, whose key is document id and value is normalized tf score of that word.
- If word is new then it is added to dictionary and if word is already exist then document is added along with normalized tf of that term . Repeat till all files are read.
- Two dictionaries are maintain , one for storing words corresponds to documents while other for storing words corresponds to index file. If word is in index dictionary then it is given more weightage than word in normal dictionary.
  W (term)= g *(tf-idf(term) in index dictionary)+(1-g)*(tf-idf(term) in document dictionary)
- If user is giving query then corresponding to each word in query we calculate term frequency and if that term is in our dictionary corresponds to both document as well as index then idf is multiply with term frequency.
  Tf-Idf(term)=tf(term)*(Idf(term) corresponds to document)
  Idf(term) =1+ log(Total number of documents / Number of documents with term in it).
- Now we have tf-idf(term) corresponds to query and normalized tf(term) corresponds to document, we will calculate the cosine similarity of term of query with each document and we get scalar value. Based on that values in descending order ,top k documents are returned.

Test Cases :

TF-IDF Score	Cosine Similarity
Query 1 : 100 animals	
Enter the Query 100 animals 447 enter the top doc 5 ('bookem.1', 0.7454048638724559) ('retrib.txt', 0.7426472343694106) ('cabin.txt', 0.6894266979492905) ('buggy.txt', 0.6877470642834582) ('kneeslapper.txt', 0.6790161217950427)	enter the top doc 5 ('contrad1.hum', 0.058026809784743935) ('bookem.1', 0.053690536851026725) ('retrib.txt', 0.05367869856584438) ('cabin.txt', 0.04754719975049403) ('buggy.txt', 0.04748175571157683)
Query 2 : one hundred animals	
Enter the Query one hundred animals 447 enter the top doc 5 ('bookem.1', 0.7454048638724559) ('retrib.txt', 0.7426472343694106) ('cabin.txt', 0.6894266979492905) ('buggy.txt', 0.6877470642834582) ('kneeslapper.txt', 0.6790161217950427)	enter the top doc 5 ('contrad1.hum', 0.03483850935520923) ('bookem.1', 0.03222140138116249) ('retrib.txt', 0.032209563095980144) ('cabin.txt', 0.02858303787152045) ('buggy.txt', 0.028517593832603248)

Observations : from the above table we can infer that both methods TF-IDF and cosine similarity will consider query “100” and “one hundred” same because in pre-processing we convert number to words so output are same.

TF-IDF Score	Cosine Similarity
Query :America hash	
Enter the Query america hash 49 enter the top doc 5 ('burltrs', 0.008007648525088035) ('mario.txt', 0.0065964315141208586) ('fea3', 0.0062983235514634745) ('breaks2.asc', 0.004854838873564861) ('frogp.txt', 0.00437266788619627)	49 enter the top doc 5 ('burltrs', 6.263674015841733e-05) ('fea3', 4.2842269014298895e-05) ('mario.txt', 2.3618067396501787e-05) ('frogp.txt', 2.192336832451053e-05) ('non4', 1.0818333262340403e-05)

Observations : From the above table we can infer that the results are different. Both the terms in query may be does not appear in index dictionary but appear in Document Dictionary. The output of both the methods are different because in TF-IDF score we only consider the tf-idf score of document term while in cosine similarity we consider the tf score of document term as well as tf-idf score of query and find the angle between query and each document in corpus. Here may be the query terms are not appearing in index dictionary so output will be only dependent on terms appear in document dictionary. We can also say that the tf-idf model will consider only term frequency, if word is frequent in document then it will return that document while cosine similarity is vector space model which try to match the whole query in document.

TF-IDF Score	Cosine Similarity
Query : Josh Renaud tooo	
Enter the Query josh renaud tooo 18 enter the top doc 5 ('sre_sei.txt', 1.113532467303391) ('sre10.txt', 1.111723975466855) ('sre_finl.txt', 1.111299055167975) ('sre08.txt', 1.1095697641416717) ('sre07.txt', 1.1095697641416717)	18 enter the top doc 5 ('sre_sei.txt', 0.25688297298689716) ('sre10.txt', 0.2568804933774863) ('sre_finl.txt', 0.2568796003772382) ('sre08.txt', 0.25687832588807674) ('sre07.txt', 0.25687832588807674)

Observation : From above table we can infer that the output of the query is same in both cases TFIDF and Cosine Similarity, it is because the term in query may be appears index dictionary so weightage is given to index dictionary as compared to document dictionary. So all the documents will be given priority which are in index dictionary.