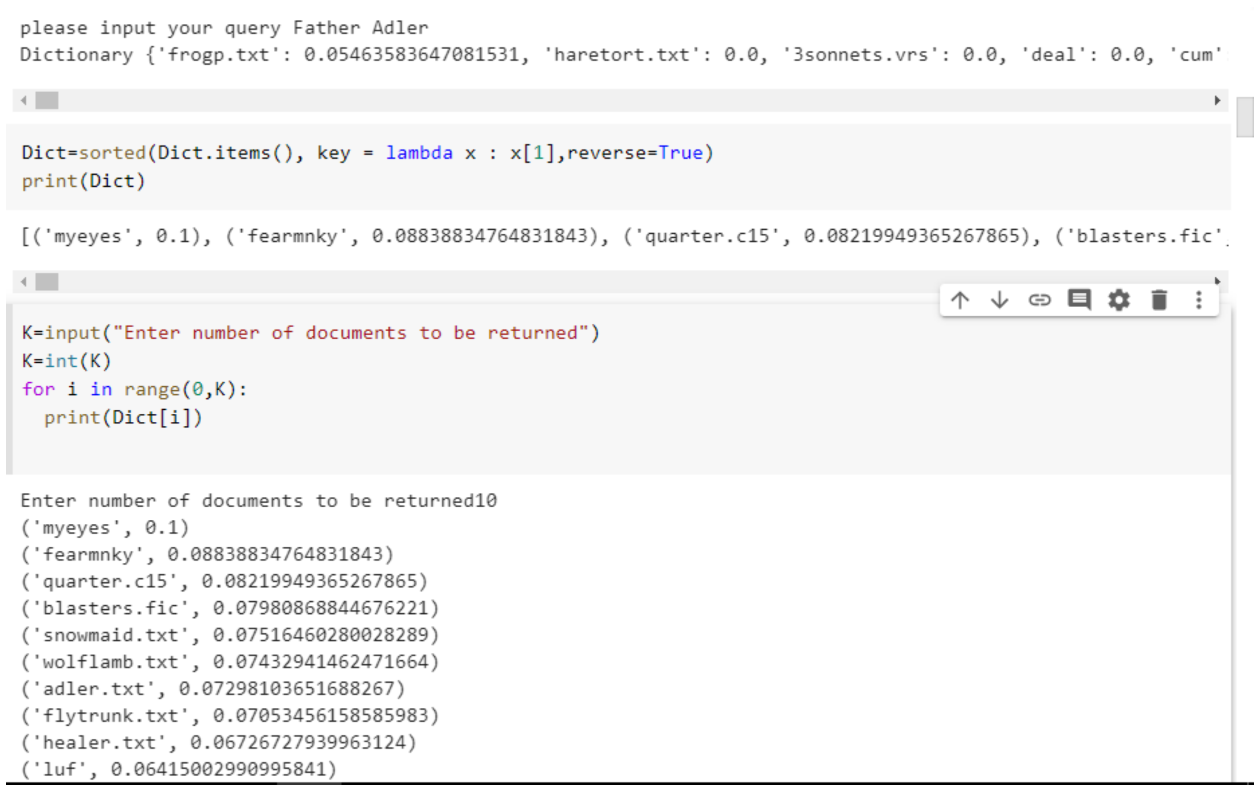
ANALYSIS

Jaccard Score :

Sample output :



```
please input your query Father Adler
Dictionary {'frogp.txt': 0.05463583647081531, 'haretort.txt': 0.0, '3sonnets.vrs': 0.0, 'deal': 0.0, 'cum'
```

```
Dict=sorted(Dict.items(), key = lambda x : x[1],reverse=True)
print(Dict)
```

```
[('myeyes', 0.1), ('fearmnky', 0.08838834764831843), ('quarter.c15', 0.08219949365267865), ('blasters.fic'
```

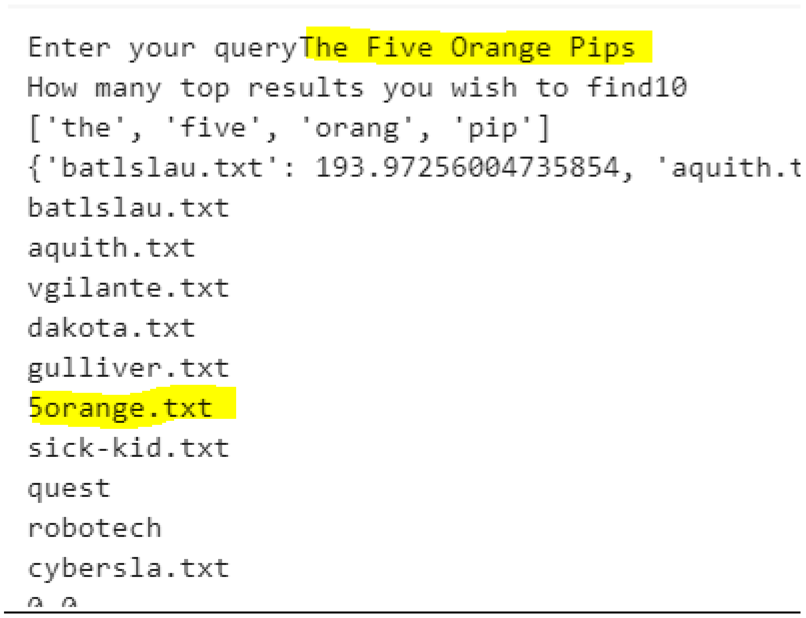
```
K=input("Enter number of documents to be returned")
K=int(K)
for i in range(0,K):
  print(Dict[i])
```

```
Enter number of documents to be returned10
('myeyes', 0.1)
('fearmnky', 0.08838834764831843)
('quarter.c15', 0.08219949365267865)
('blasters.fic', 0.07980868844676221)
('snowmaid.txt', 0.07516460280028289)
('wolflamb.txt', 0.07432941462471664)
('adler.txt', 0.07298103651688267)
('flytrunk.txt', 0.07053456158585983)
('healer.txt', 0.06726727939963124)
('luf', 0.06415002990995841)
```

TF IDF Variants

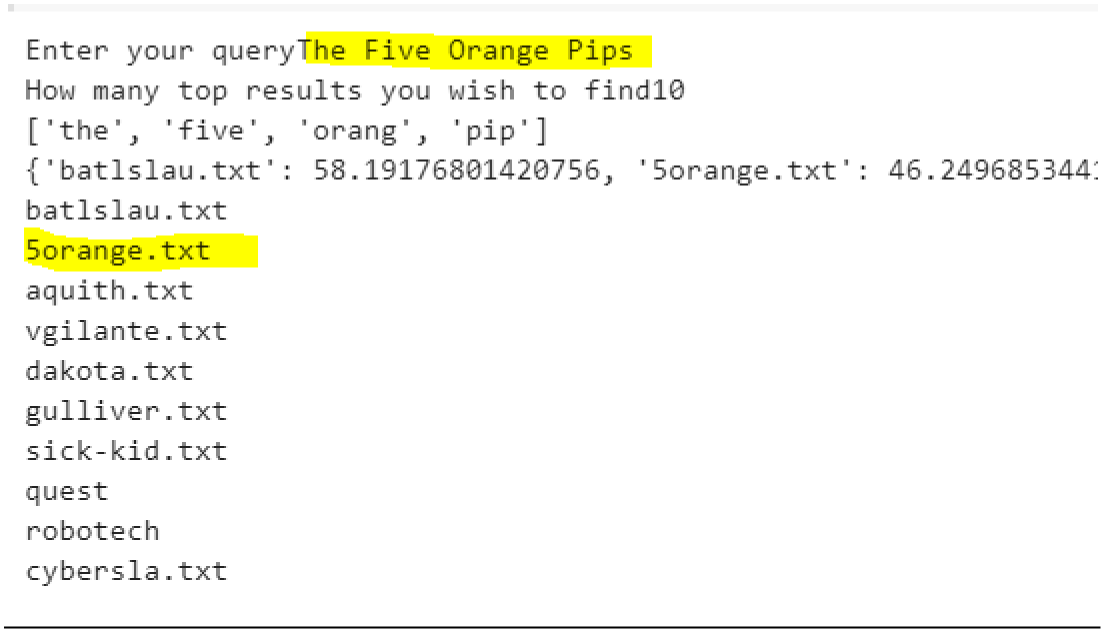
TF score	**IDF Score**
n (natural Term frequency)	Log(N/df)

Output without handling title :



```
Enter your queryThe Five Orange Pips
How many top results you wish to find10
['the', 'five', 'orang', 'pip']
{'batlslau.txt': 193.97256004735854, 'aquith.t
batlslau.txt
aquith.txt
vgilante.txt
dakota.txt
gulliver.txt
5orange.txt
sick-kid.txt
quest
robotech
cybersla.txt
0.0
```

Output with handling title :

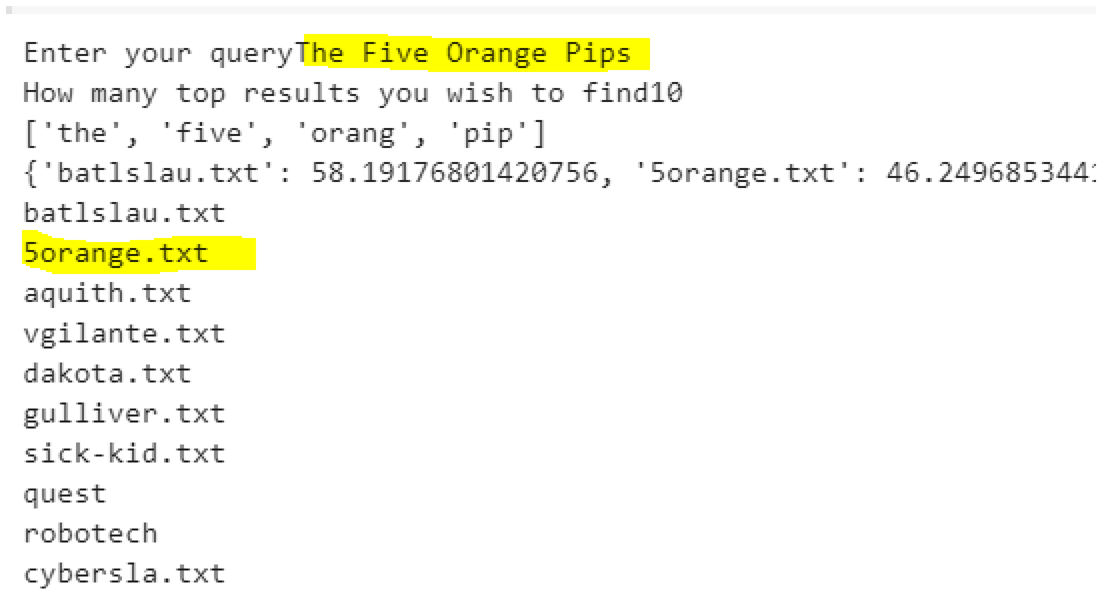


```
Enter your queryThe Five Orange Pips
How many top results you wish to find10
['the', 'five', 'orang', 'pip']
{'batlslau.txt': 58.19176801420756, '5orange.txt': 46.249685344
batlslau.txt
5orange.txt
aquith.txt
vgilante.txt
dakota.txt
gulliver.txt
sick-kid.txt
quest
robotech
cybersla.txt
```

From the above output we can clearly note that handling title is giving better result to more relevant documents

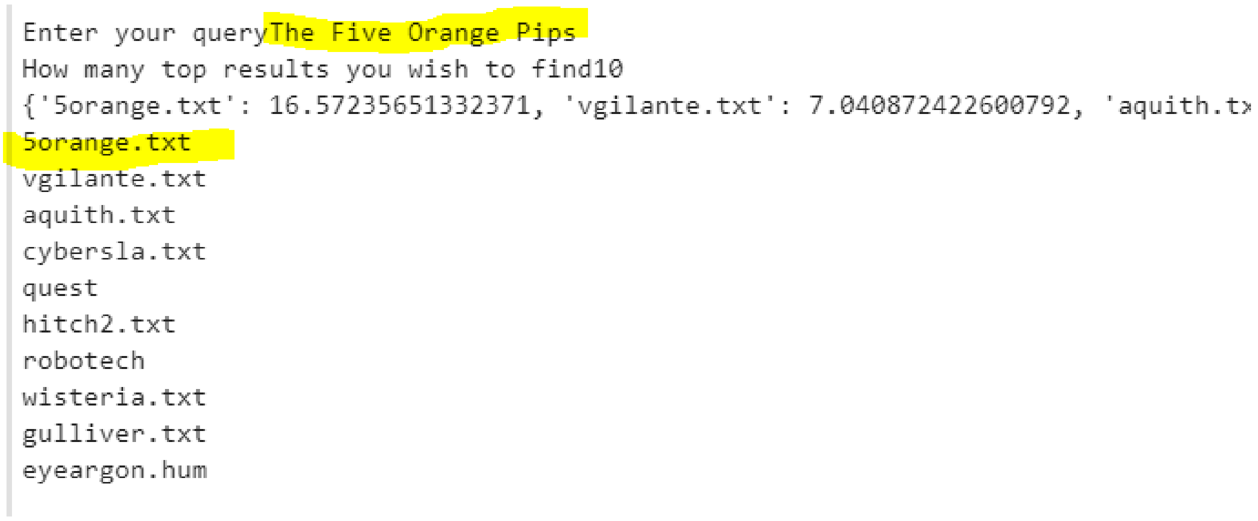
TF score	IDF Score
Logarithmic	Log(N/df)

Output without handling title :



```
Enter your queryThe Five Orange Pips
How many top results you wish to find10
['the', 'five', 'orang', 'pip']
{'batlslau.txt': 58.19176801420756, '5orange.txt': 46.249685344
batlslau.txt
5orange.txt
aquith.txt
vgilante.txt
dakota.txt
gulliver.txt
sick-kid.txt
quest
robotech
cybersla.txt
```

Output with handling title :

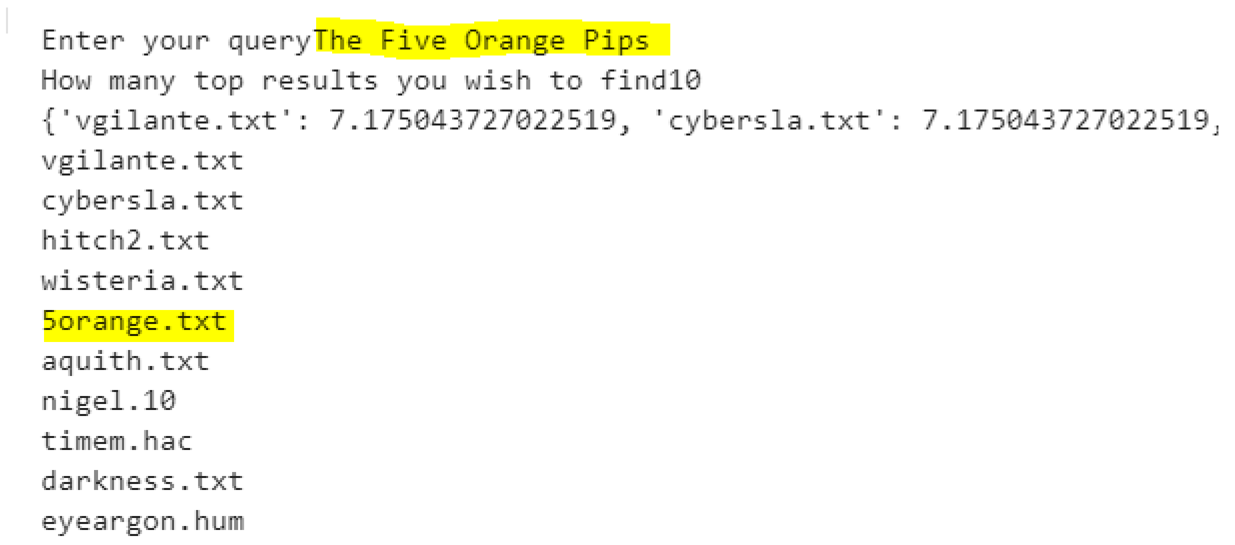


```
Enter your queryThe Five Orange Pips
How many top results you wish to find10
{'5orange.txt': 16.57235651332371, 'vgilante.txt': 7.040872422600792, 'aquith.tx
5orange.txt
vgilante.txt
aquith.txt
cybersla.txt
quest
hitch2.txt
robotech
wisteria.txt
gulliver.txt
eyeargon.hum
```

From the above two outputs we can see that handling title is giving more rank to relevant documents.

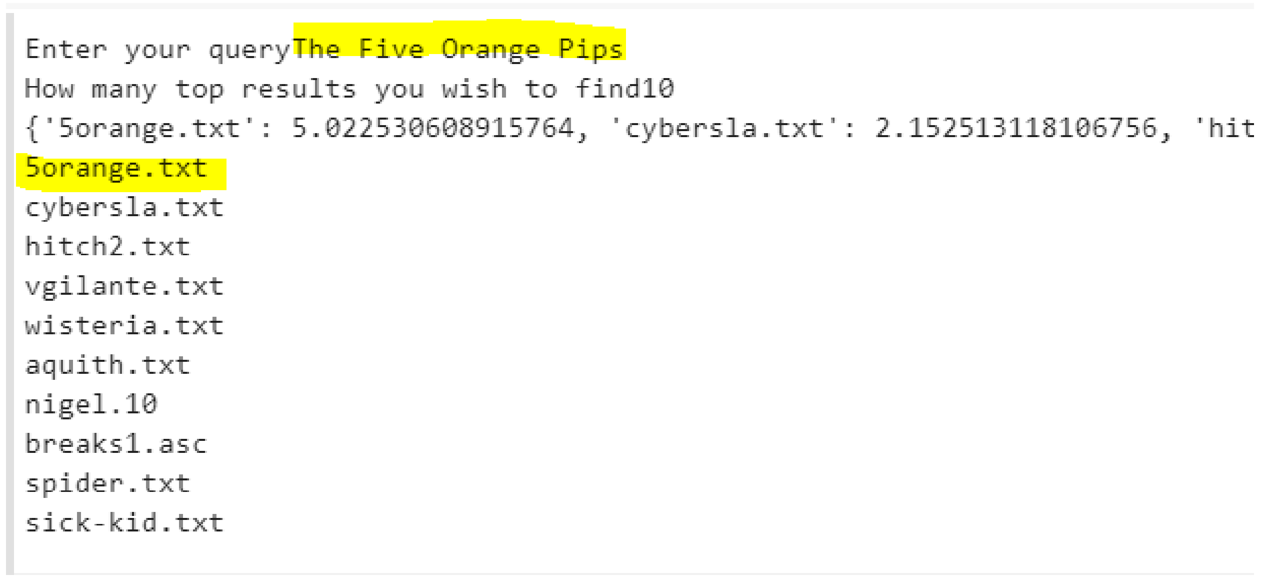
TF score	IDF Score
{0,1}(Boolean)	Log(N/df)

Output without handling title :



```
Enter your queryThe Five Orange Pips
How many top results you wish to find10
{'vgilante.txt': 7.175043727022519, 'cybersla.txt': 7.175043727022519,
vgilante.txt
cybersla.txt
hitch2.txt
wisteria.txt
5orange.txt
aquith.txt
nigel.10
timem.hac
darkness.txt
eyeargon.hum
```

Output with handling title :



```
Enter your queryThe Five Orange Pips
How many top results you wish to find10
{'5orange.txt': 5.022530608915764, 'cybersla.txt': 2.152513118106756, 'hit
5orange.txt
cybersla.txt
hitch2.txt
vgilante.txt
wisteria.txt
aquith.txt
nigel.10
breaks1.asc
spider.txt
sick-kid.txt
```

Comparison Among various TF-IDF variants

On the basis of above analysis done on all the three flavors of tf-idf scoring we can conclude that variant 2 performs best followed by variant 1 and lastly variant 3.

Logarithmic	Log(N/df)

n (natural Term frequency)	Log(N/df)

{0,1}(Boolean)	Log(N/df)

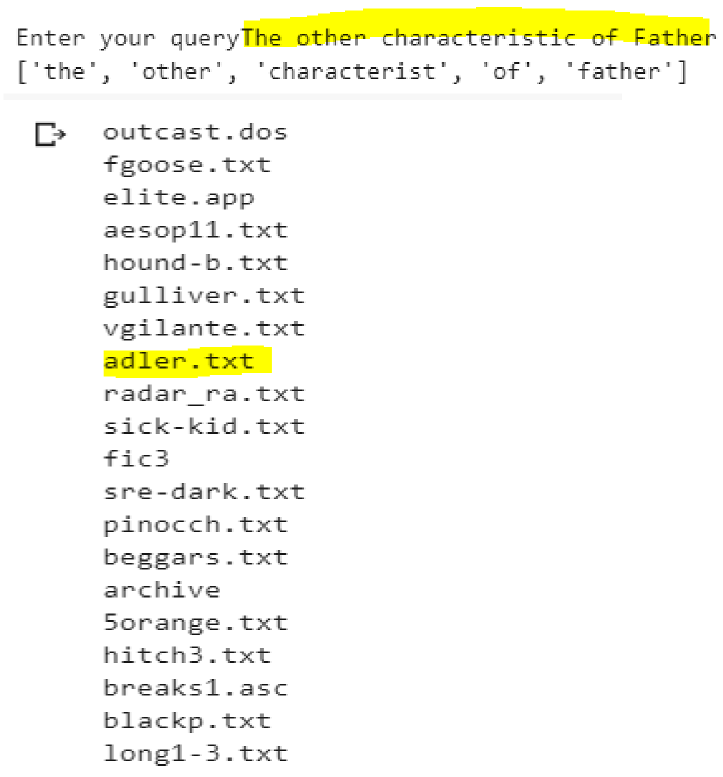
Please note that I have fixed IDF calculation.

Now the performance will totally depend upon the various flavors of TF .

The reason why logarithmic tf performs best is that log factor becomes asymptote to a line parallel to y axis after some time and doesn't increase proportionally with increase in tf value.

Cosine Similarity

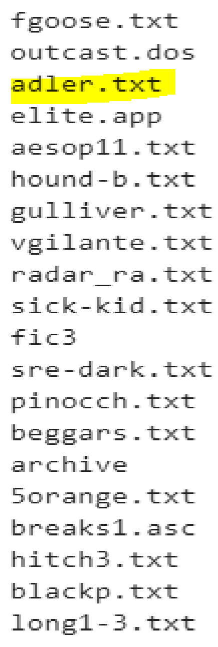
Sample output without title handling



```
Enter your queryThe other characteristic of Father
['the', 'other', 'characterist', 'of', 'father']

    outcast.dos
    fgoose.txt
    elite.app
    aesop11.txt
    hound-b.txt
    gulliver.txt
    vgilante.txt
    adler.txt
    radar_ra.txt
    sick-kid.txt
    fic3
    sre-dark.txt
    pinocch.txt
    beggars.txt
    archive
    5orange.txt
    hitch3.txt
    breaks1.asc
    blackp.txt
    long1-3.txt
```

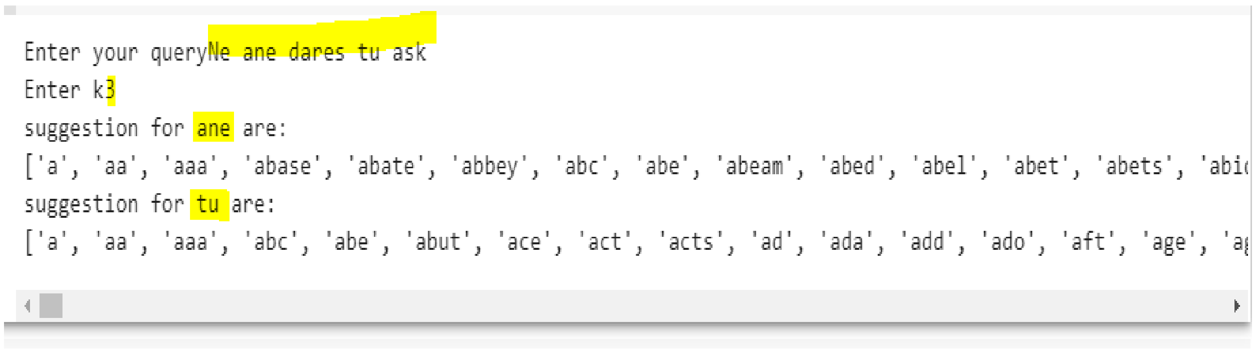
<u>Sample output with title handling</u>



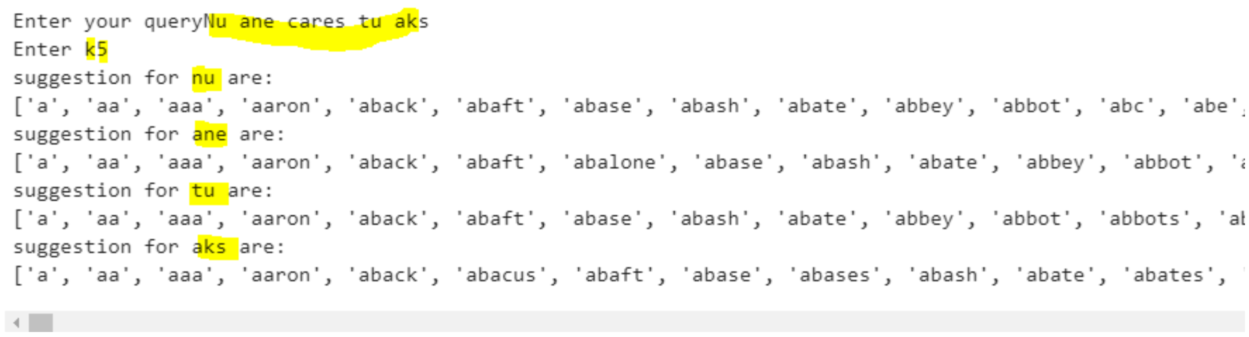
```
fgoose.txt
outcast.dos
adler.txt
elite.app
aesop11.txt
hound-b.txt
gulliver.txt
vgilante.txt
radar_ra.txt
sick-kid.txt
fic3
sre-dark.txt
pinocch.txt
beggars.txt
archive
5orange.txt
breaks1.asc
hitch3.txt
blackp.txt
long1-3.txt
```

Question 2 :

Sample Output



```
Enter your queryNe ane dares tu ask
Enter k3
suggestion for ane are:
['a', 'aa', 'aaa', 'abase', 'abate', 'abbey', 'abc', 'abe', 'abeam', 'abed', 'abel', 'abet', 'abets', 'abi
suggestion for tu are:
['a', 'aa', 'aaa', 'abc', 'abe', 'abut', 'ace', 'act', 'acts', 'ad', 'ada', 'add', 'ado', 'aft', 'age', 'a
```



```
Enter your queryNu ane cares tu aks
Enter k5
suggestion for nu are:
['a', 'aa', 'aaa', 'aaron', 'aback', 'abaft', 'abase', 'abash', 'abate', 'abbey', 'abbot', 'abc', 'abe',
suggestion for ane are:
['a', 'aa', 'aaa', 'aaron', 'aback', 'abaft', 'abalone', 'abase', 'abash', 'abate', 'abbey', 'abbot', '
suggestion for tu are:
['a', 'aa', 'aaa', 'aaron', 'aback', 'abaft', 'abase', 'abash', 'abate', 'abbey', 'abbot', 'abbots', 'a
suggestion for aks are:
['a', 'aa', 'aaa', 'aaron', 'aback', 'abacus', 'abaft', 'abase', 'abases', 'abash', 'abate', 'abates',
```

Comparison Among various implementations

Jaccard score gives poor performance for this dataset. The reason can be as follows:

The denominator this score depends upon the length of the document. For this corpus length of the docs is varying highly which gives rise to wrong results and increase false positives.

I have tried to normalize denominator by square root factor.It is giving comparatively better results but still not very satisfactory.

TF-IDF works better than the jaccard score. Although it is one heuristic but works well in most of the scenarios and end up giving lesser number of false positives in the top k results.

TF-IDF score along with the weighted scheme of the title even performs more good.

Cosine similarity has overall best performance among all three variants. It normalizes the query and document vector and depends upon the angle between them not on the distance. This approach makes length of the vector less dominating.

If we think of storing the document vector for all the terms in the token then this takes a lot of space. Another problem is this is highly sparse. Therefore I have calculated the document vector for only query terms at run time.