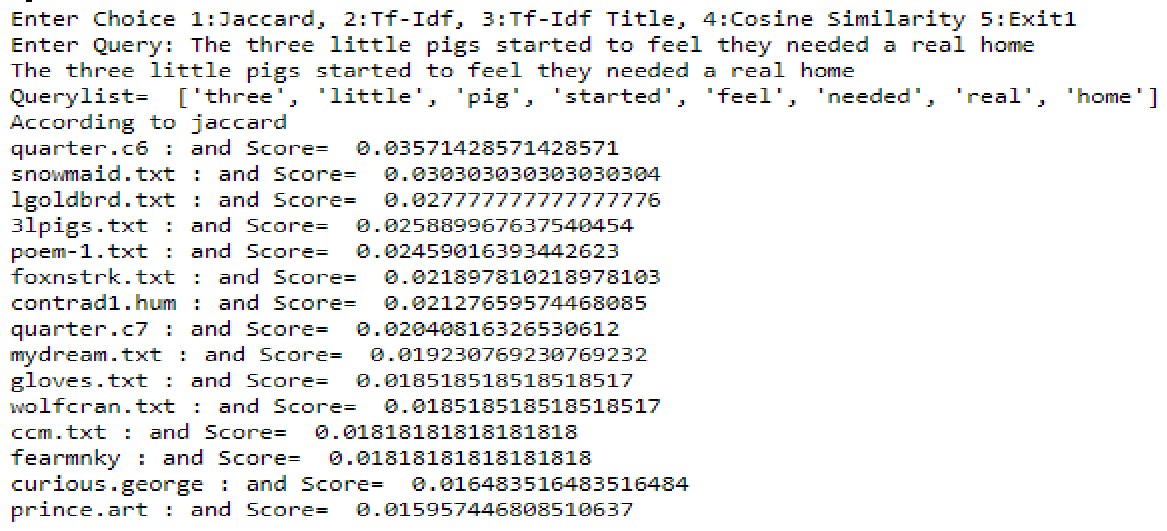
Analysis(Q2)

Akshaj(MT19111)

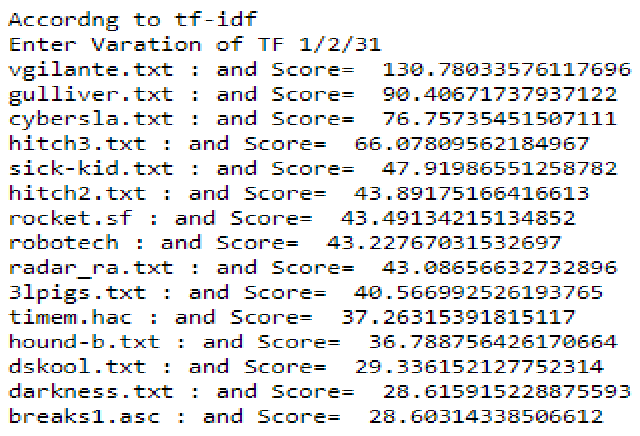
1:]Jaccard



```
Enter Choice 1:Jaccard, 2:Tf-Idf, 3:Tf-Idf Title, 4:Cosine Similarity 5:Exit1
Enter Query: The three little pigs started to feel they needed a real home
The three little pigs started to feel they needed a real home
Querylist=  ['three', 'little', 'pig', 'started', 'feel', 'needed', 'real', 'home']
According to jaccard
quarter.c6 : and Score=  0.03571428571428571
snowmaid.txt : and Score=  0.030303030303030304
lgoldbrd.txt : and Score=  0.027777777777777776
3lpigs.txt : and Score=  0.025889967637540454
poem-1.txt : and Score=  0.02459016393442623
foxnstrk.txt : and Score=  0.021897810218978103
contrad1.hum : and Score=  0.02127659574468085
quarter.c7 : and Score=  0.02040816326530612
mydream.txt : and Score=  0.019230769230769232
gloves.txt : and Score=  0.018518518518518517
wolfcran.txt : and Score=  0.018518518518518517
ccm.txt : and Score=  0.01818181818181818
fearmnky : and Score=  0.01818181818181818
curious.george : and Score=  0.016483516483516484
prince.art : and Score=  0.015957446808510637
```

Here the above query is from 3lpigs.txt and 3lpigs.txt is on 4th position. Jaccard coefficient is easy to implement. If there is no common elements then Jaccard coefficient gives zero. Jaccard coefficient also does not consider context. Jaccard coefficient can give more preference to more frequent words in the document. Jaccard coefficient does not provide special score to the document in which term exists in title.

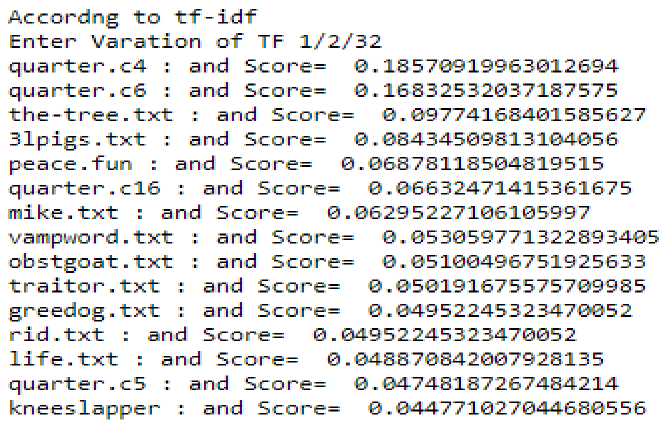
2:]tf-idf (With raw tf)



```
Accordng to tf-idf
Enter Varation of TF 1/2/31
vgilante.txt : and Score=  130.78033576117696
gulliver.txt : and Score=  90.40671737937122
cybersla.txt : and Score=  76.75735451507111
hitch3.txt : and Score=  66.07809562184967
sick-kid.txt : and Score=  47.91986551258782
hitch2.txt : and Score=  43.89175166416613
rocket.sf : and Score=  43.49134215134852
robotech : and Score=  43.22767031532697
radar_ra.txt : and Score=  43.08656632732896
3lpigs.txt : and Score=  40.566992526193765
timem.hac : and Score=  37.26315391815117
hound-b.txt : and Score=  36.788756426170664
dskool.txt : and Score=  29.336152127752314
darkness.txt : and Score=  28.615915228875593
breaks1.asc : and Score=  28.60314338506612
```

Here the above query is from 3lpigs.txt and 3lpigs.txt is on 10th position because here “tf” is not normalized and if length of doc is more then there are high chances that term frequency is high. So it will give false results. Here this method considers just raw tf so here if document length is more then this method will give preference to those documents. Here if valid query is present in document will small length then too it can be ranked later.

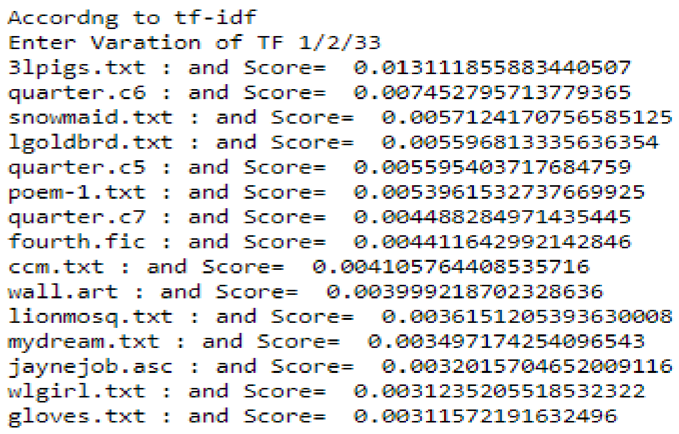
3:]tf-idf(tf normalized with length of document)



```
Accordng to tf-idf
Enter Varation of TF 1/2/32
quarter.c4 : and Score=  0.18570919963012694
quarter.c6 : and Score=  0.16832532037187575
the-tree.txt : and Score=  0.09774168401585627
3lpigs.txt : and Score=  0.08434509813104056
peace.fun : and Score=  0.06878118504819515
quarter.c16 : and Score=  0.06632471415361675
mike.txt : and Score=  0.06295227106105997
vampword.txt : and Score=  0.053059771322893405
obstgoat.txt : and Score=  0.05100496751925633
traitor.txt : and Score=  0.050191675575709985
greedog.txt : and Score=  0.04952245323470052
rid.txt : and Score=  0.04952245323470052
life.txt : and Score=  0.048870842007928135
quarter.c5 : and Score=  0.04748187267484214
kneeslapper : and Score=  0.044771027044680556
```

Here the above query is from 3lpigs.txt and 3lpigs.txt is on 4th position. Here “tf” is normalized with length of document so if document is large then frequency of term in it will be more, so after normalizing it by length this mothed will not give more preference to large documents. So here ranking will be done more efficiently than non normalized one.

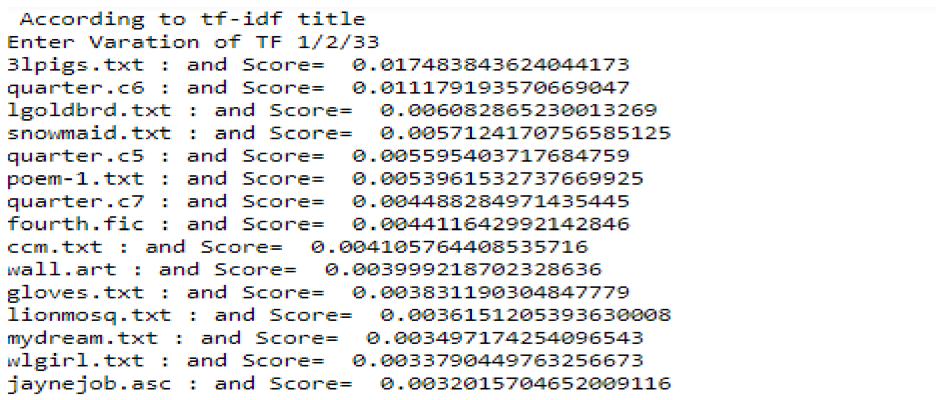
4:]tf-idf(tf normalized with log)



```
Accordng to tf-idf
Enter Varation of TF 1/2/33
3lpigs.txt : and Score=  0.013111855883440507
quarter.c6 : and Score=  0.007452795713779365
snowmaid.txt : and Score=  0.0057124170756585125
1goldbrd.txt : and Score=  0.005596813335636354
quarter.c5 : and Score=  0.005595403717684759
poem-1.txt : and Score=  0.0053961532737669925
quarter.c7 : and Score=  0.004488284971435445
fourth.fic : and Score=  0.004411642992142846
ccm.txt : and Score=  0.004105764408535716
wall.art : and Score=  0.003999218702328636
lionmosq.txt : and Score=  0.0036151205393630008
mydream.txt : and Score=  0.003497174254096543
jaynejob.asc : and Score=  0.0032015704652009116
wlgirl.txt : and Score=  0.0031235205518532322
gloves.txt : and Score=  0.00311572191632496
```

Here the above query is from 3lpigs.txt and 3lpigs.txt is on 1th position. So compared to above 3 methods this method performs well. Here “tf” is normalized with “log” and also is divided by size of document so this method does not consider large documents as more relevant.

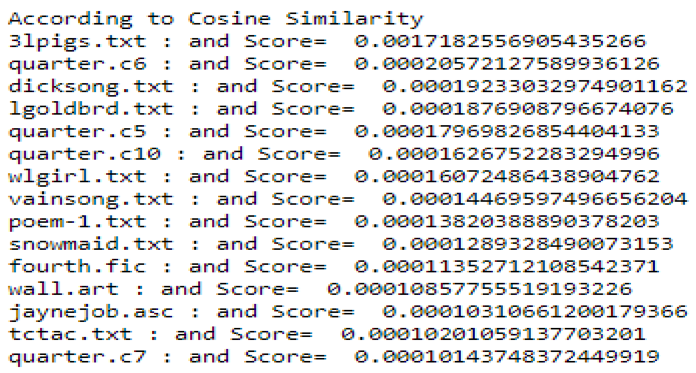
5:]tf-idf(Considering title)



```
 According to tf-idf title
Enter Varation of TF 1/2/33
3lpigs.txt : and Score=  0.017483843624044173
quarter.c6 : and Score=  0.011179193570669047
lgoldbrd.txt : and Score=  0.006082865230013269
snowmaid.txt : and Score=  0.0057124170756585125
quarter.c5 : and Score=  0.005595403717684759
poem-1.txt : and Score=  0.005396153273766992 5
quarter.c7 : and Score=  0.004488284971435445
fourth.fic : and Score=  0.004411642992142846
ccm.txt : and Score=  0.004105764408535716
wall.art : and Score=  0.003999218702328636
gloves.txt : and Score=  0.003831190304847779
lionmosq.txt : and Score=  0.0036151205393630008
mydream.txt : and Score=  0.003497174254096543
wlgirl.txt : and Score=  0.003379044976325667 3
jaynejob.asc : and Score=  0.0032015704652009116
```

Here the above query is from 3lpigs.txt and 3lpigs.txt is on 1th position. Considering title this method gives correct result. Here in this method title is consider so if query term is present in title. So here more relevant documents will be ranked first. But in this case if query is relevant to doc1 but not present in doc1 title and some of the query terms are present another documents title then it will give more weightage to other document.

6:]Cosine Similarity

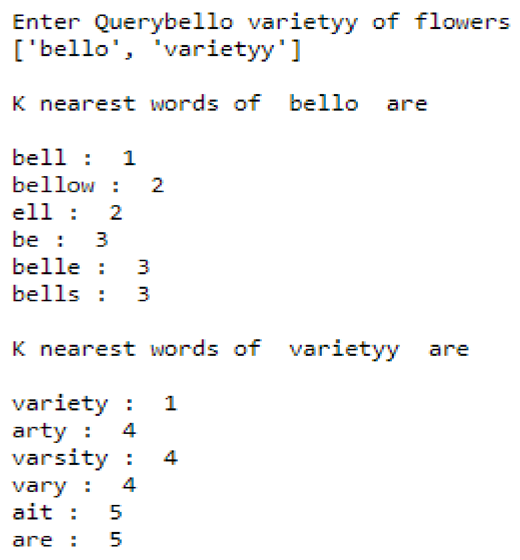


```
According to Cosine Similarity
3lpigs.txt : and Score=  0.0017182556905435266
quarter.c6 : and Score=  0.00020572127589936126
dicksong.txt : and Score=  0.00019233032974901162
lgoldbrd.txt : and Score=  0.0001876908796674076
quarter.c5 : and Score=  0.0001796982685440413 3
quarter.c10 : and Score=  0.0001626752283294996
wlgirl.txt : and Score=  0.00016072486438904762
vainsong.txt : and Score=  0.00014469597496656204
poem-1.txt : and Score=  0.0001382038889037820 3
snowmaid.txt : and Score=  0.0001289328490073153
fourth.fic : and Score=  0.0001135271210854237 1
wall.art : and Score=  0.00010857755519193226
jaynejob.asc : and Score=  0.00010310661200179366
tctac.txt : and Score=  0.00010201059137703201
quarter.c7 : and Score=  0.00010143748372449919
```

Here the above query is from 3lpigs.txt and 3lpigs.txt is on 1th position. Cosine similarity works good as it ranked 3lpigs.txt first. Cosine similarity checks similarity between document and query so as compare to other methods cosine similarity performs well. But cosine similarity is some what difficult to implement.

Considering all the above results we can say that "tf" normalized with log performs good. And tf-idf considering title also works good in some cases and if in case query words are not from title then it can give preference to another document where single query word present in that title. Cosine similarity works best in all above methods.

Q2:] Edit-Distance.



```
Enter Querybello varietyy of flowers
['bello', 'varietyy']

K nearest words of  bello  are

bell :  1
bellow :  2
ell :  2
be :  3
belle :  3
bells :  3

K nearest words of  varietyy  are

variety :  1
arty :  4
varsity :  4
vary :  4
ait :  5
are :  5
```

Here miss-spelled words in query are “bello” and “varietyy” so 5 nearest words of “bello” are shown above with cost for converting above word to “bello” and same goes for “variety”.