2014Fall7646 Project 1C

Name: Akshat Harit
GtID: 903090915

Correct Answer
The answer matches the one on the forums. However it is not correct in the sense that files aren't classified correctly. This could be due to a small data set and tf-idf not being a sufficient method to interpret meaning from the words.

Metric to assess results of classification
The metric designed is a simple one that just outputs the percentage of correct classification as a percentage of total items classified. The initial(tf-idf with cosine similarity) scheme gives 56.25% accuracy for the tf-idf classifier.

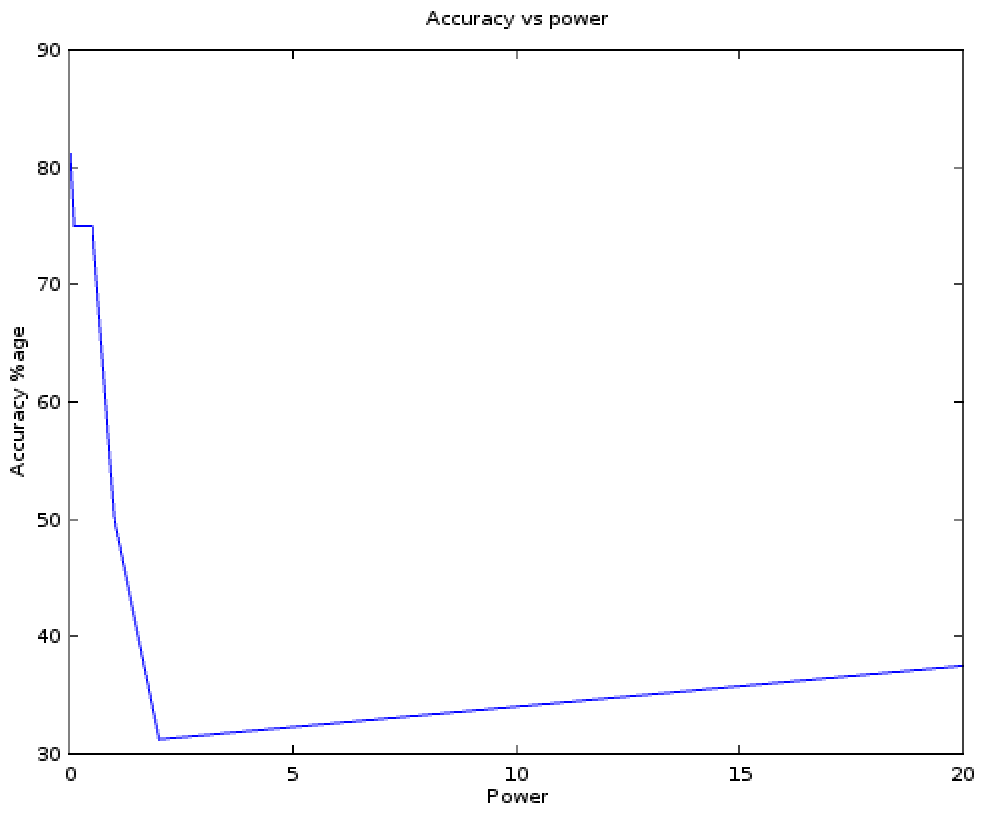
Improvement(Voting Scheme and alternative idf function)
I used a voting mechanism to improve the results of classifier.
The rational behind the method was that a good file might be best matched to a bad file, but over the set of all the files, a good file should match more to good files than bad files. So even if there is a single anomaly i.e dissimilar files matching, if we give votes to each of the files, the aggregate should give a better classification.
The method proposed was as follows. A classification number was associated with each file. As the file was compared with other files, the similarity number that was returned(in this case cosine similarity) was added to the classification with an assigned weight. The weight was the 1/total number of files in that category. This is done to ensure that if there are disproportionate number of good or bad files in the mix, one set doesn't dominate others based only on numerical superiority. Furthermore bad files were given negative weights and good files were given positive weights. The weights were added and depending on whether the final result was positive or negative, the file was classified as good or bad.
This did not lead to improvement. In fact accuracy worsened to 43.75% Possible reasons are that a single value overwhelms the classification. For example for good02.txt(data output in this format has been commented out, so actual output for partC.py doesn't give the following information)

```
Fraternizing with good files,good01.txt advantage is 0.0188528813909
Fraternizing with good files,good03.txt advantage is 0.036329864994
Fraternizing with good files,good04.txt advantage is 0.0141798464213
Fraternizing with good files,good05.txt advantage is 0.0352862046767
Fraternizing with good files,good06.txt advantage is 0.0206233053098
Fraternizing with good files,good07.txt advantage is 0.0144768363541
Fraternizing with good files,good08.txt advantage is 0.0133542946721
Fraternizing with bad files+bad01.txt, penalty is -0.0219637347117
Fraternizing with bad files+bad02.txt, penalty is -0.0150676269247
Fraternizing with bad files+bad03.txt, penalty is -0.217397767126
Fraternizing with bad files+bad04.txt, penalty is -0.0255991728669
Fraternizing with bad files+bad05.txt, penalty is -0.0239329925953
Fraternizing with bad files+bad06.txt, penalty is -0.0285863831858
Fraternizing with bad files+bad07.txt, penalty is -0.024558358699
Fraternizing with bad files+bad08.txt, penalty is -0.0233989601808
No of files(good)     Classification,      No of files(Bad)          Bad Classification
7                     0.153103233819       8                         -0.380504996291
good02.txt is bad File with confidence of -0.0256912339908
```

For good02.txt, a single cosine similarity with bad03.txt(bold), which is an order of magnitude larger than other cosine similarity, overwhelms the classification. Also there is not much difference in the value of a good file's cosine similarity with good or bad files. Also the data set is small i.e only 16 files. A single mismatch results in very large change to the accuracy.

As an aside, replacing idf log by a power function with with very small coefficients seems to improve efficieny quite a lot. I have left a commented out a line in file partB.py that can do this. For the line search for text #POWER and comment out the following line. The following table and graph demonstrates this

Power	Classifier Efficiency
20	37.5%
2	31.25%
1	50%
0.5	75%
0.1	75%
0.01	81.25%
0.001	81.25%



I have no concrete theory right now as to why this is happening. It could be because at high values of number of files/occurrence the power method gives lower result than logarithm. This would decrease further the contribution of idf at high values while still maintaining it a higher value. So a word that occurs in say two documents would still have higher weight. Interestingly, using a voting scheme with combined with alternative idf function actually worsened the results.

Comparison

As compared to the base solution, the proposed voting scheme performs worse while the alternative idf functon performs much better with accuracy achieving 81.25 %. In fact the voting scheme performs worse than random output. However, I think a more comprehensive test with much larger data set is needed to completely evaluate the approaches. I think that as the data set size increases, the voting method would perform better as it is insulated against a single or a smaller number of mis-classifications as compared to total data.

RESULTS
partA.py

```
filename, match, cosine
good01.txt, bad08.txt,0.0463359148237
good02.txt, bad03.txt,0.217397767126
good03.txt, good02.txt,0.036329864994
good04.txt, bad06.txt,0.0370217425463
good05.txt, good01.txt,0.0374741642762
good06.txt, bad04.txt,0.0441460154874
good07.txt, good08.txt,0.0853077709574
good08.txt, bad07.txt,0.103574391881
bad01.txt, bad08.txt,0.0508196797044
bad02.txt, bad06.txt,0.0758642113122
bad03.txt, good02.txt,0.217397767126
bad04.txt, bad06.txt,0.0494963642645
bad05.txt, bad03.txt,0.0551295103495
bad06.txt, bad08.txt,0.110651442251
bad07.txt, good08.txt,0.103574391881
bad08.txt, bad06.txt,0.110651442251
```

partB.py

```
filename, match, cosine
bad01.txt,bad08.txt,0.0508196797044
bad02.txt,bad06.txt,0.0758642113122
bad03.txt,good02.txt,0.217397767126
bad04.txt,bad06.txt,0.0494963642645
bad05.txt,bad03.txt,0.0551295103495
bad06.txt,bad08.txt,0.110651442251
bad07.txt,good08.txt,0.103574391881
bad08.txt,bad06.txt,0.110651442251
good01.txt,bad08.txt,0.0463359148237
good02.txt,bad03.txt,0.217397767126
good03.txt,good02.txt,0.036329864994
good04.txt,bad06.txt,0.0370217425463
good05.txt,good01.txt,0.0374741642762
good06.txt,bad04.txt,0.0441460154874
good07.txt,good08.txt,0.0853077709574
good08.txt,bad07.txt,0.103574391881
overall score: 56.25%
```

partC.py

```
good01.txt is bad File with confidence of 0.00308617595299
good02.txt is bad File with confidence of 0.0256912339908
good03.txt is good File with confidence of 0.000288002534027
good04.txt is bad File with confidence of 0.00645390928353
good05.txt is bad File with confidence of 0.00135551055546
good06.txt is bad File with confidence of 0.00301620804591
good07.txt is bad File with confidence of 0.00308360890778
good08.txt is bad File with confidence of 0.00779024759806
```

```
bad01.txt is bad File with confidence of 0.00896699194206
bad02.txt is bad File with confidence of 0.0079929976106
bad03.txt is good File with confidence of 0.0158581144435
bad04.txt is bad File with confidence of 0.00691240257632
bad05.txt is bad File with confidence of 0.0103742855802
bad06.txt is bad File with confidence of 0.0280846205004
bad07.txt is good File with confidence of 0.00518220030585
bad08.txt is bad File with confidence of 0.0203790892334
overall score: 43.75%
```

Alternative idf function change to power with .01 as power(Look into file partB.py and comment out the line following #POWER)

```
filename, match, cosine
good01.txt,good05.txt,0.675610005043
good02.txt,good05.txt,0.738562291893
good03.txt,good02.txt,0.630191273585
good04.txt,good07.txt,0.744435221077
good05.txt,good02.txt,0.738562291893
good06.txt,good05.txt,0.70103280337
good07.txt,good08.txt,0.765564739079
good08.txt,good07.txt,0.765564739079
bad01.txt,bad08.txt,0.746036769653
bad02.txt,good04.txt,0.625559991581
bad03.txt,good02.txt,0.725503632976
bad04.txt,bad05.txt,0.730354541204
bad05.txt,bad04.txt,0.730354541204
bad06.txt,bad08.txt,0.789702181415
bad07.txt,good07.txt,0.73651133558
bad08.txt,bad06.txt,0.789702181415
overall score: 81.25%
```