CSA E0 226: Linear Algebra and Probability (25th November 2018)

Assignment 2

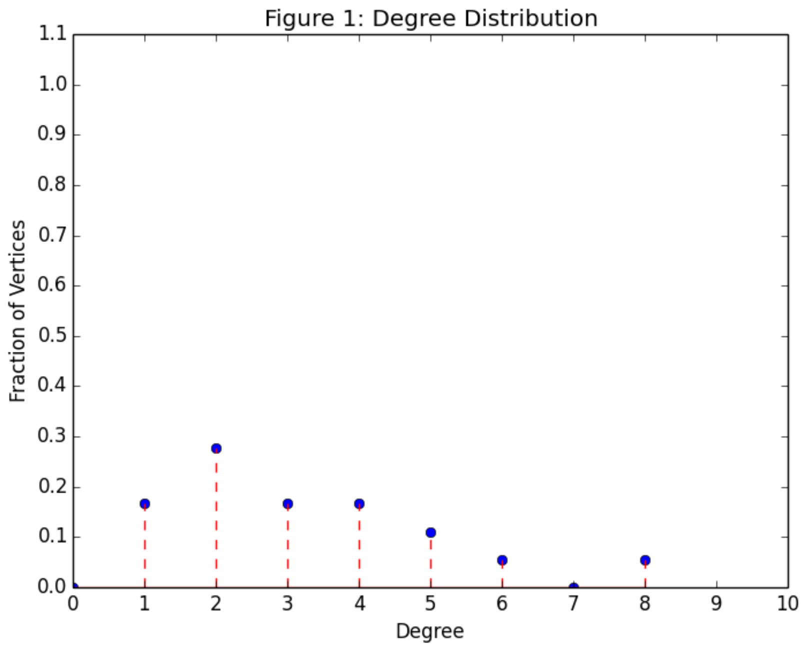
Submitted by:- YASH KHANNA [MTech (Res.) CSA / 04-04-00-10-22-18-1-15563]

- Used **Python 2.7** for the assignment (Both the problems)
- In problem 2, I have fixed the value of **K = 3** and **M = 250**

Problem 1

Task - 1
Write code to obtain the degree distribution of the nodes. Plot it. What can you infer from the distribution? Does it follow any familiar distribution?

Solution: The given network is quite small, so it is hard to find a particular distribution. However, the following facts are well known. Given a network on N vertices and the probability of an edge being present in the graph be p (Random Graph), then the degrees roughly follow the poisson distribution. Also for scale-free networks which have a very uneven distribution of connections. Some nodes have very high degrees of connectivity (hubs), while most have small degrees. Then the degrees follow a power-law distribution. (A mathematical relation that forms linear plots when data is transformed into log-log coordinates). The plot and the data are given below,



Degree Distribution :
Node 0 (Joanna) : 1
Node 1 (Tyrion) : 3
Node 2 (Jaime) : 4
Node 3 (Cersei) : 5
Node 4 (Robert) : 2
Node 5 (Varys) : 4
Node 6 (Tywin) : 6
Node 7 (Joffrey) : 2
Node 8 (Sansa) : 2
Node 9 (Katelyn) : 5
Node 10 (Ned) : 8
Node 11 (Jon) : 2
Node 12 (Arya) : 4
Node 13 (Bran) : 3
Node 14 (Benjamin) : 1
Node 15 (Hodor) : 1
Node 16 (Rob) : 2
Node 17 (Little-finger) : 3

Task - 2
Calculate the node centrality using any one from the different standard node centrality measures. Plot a histogram, taking uniform interval from the entire range of centrality values obtained. Name the top two central nodes of the given graph.

Solution:
Central Node #1 (Highest) : Node 10 (Ned)
Central Node #2 (Second-Highest) : Node 6 (Tywin)

I have used the Harmonic Centrality Measure between nodes.
`https://en.wikipedia.org/wiki/Centrality#Harmonic_centrality`

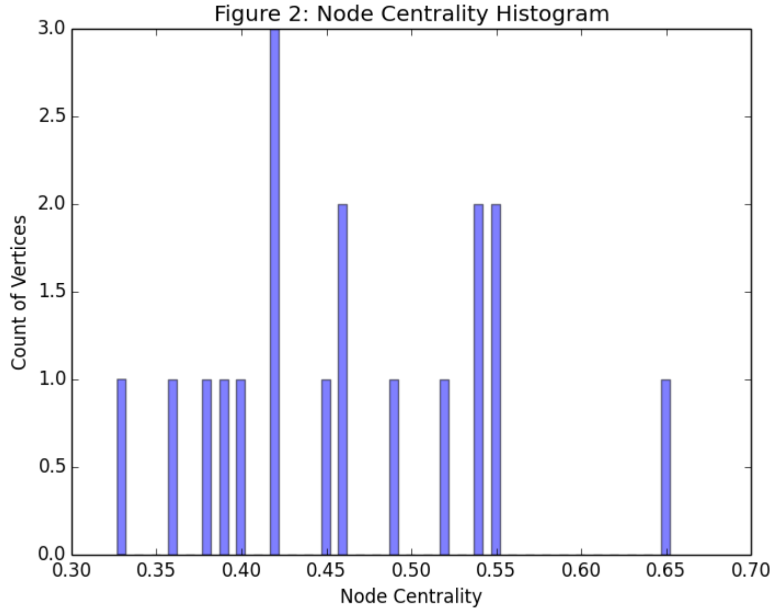
Task - 3
Find the edge centrality using any centrality measure for all edges. Which edge is most central?

Solution:
The most central edge is between : Node 5 (Varys) and Node 17 (Little-finger)

I have used the Betweenness Centrality Measure between edges.
`https://en.wikipedia.org/wiki/Betweenness_centrality`

Task - 4
Create the adjacency matrix of the given network. Find out the Laplacian matrix and

Figure 2: Node Centrality Histogram



normalized Laplacian matrix. Calculate the eigenvalues and eigen- vectors of the Laplcian. You are not allowed to use any library function to find the eigenvalues and eigenvectors. You can only use libraries to find the roots of a polynomial equation (one option is to use numpy.roots). Check if the eigenvalues are real or not.

Solution:
Adjacency Matrix :

```
0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 1.0 1.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 1.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0 1.0 1.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 1.0 1.0 1.0 1.0 0.0 1.0 1.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 1.0 0.0 1.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 1.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

```
Laplacian Matrix :
1.0 0.0 0.0 0.0 0.0 0.0 -1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 3.0 -1.0 0.0 0.0 -1.0 -1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 -1.0 4.0 -1.0 0.0 0.0 -1.0 -1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 -1.0 5.0 -1.0 -1.0 -1.0 -1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 -1.0 2.0 0.0 -1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 -1.0 0.0 -1.0 0.0 4.0 -1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.0
-1.0 -1.0 -1.0 -1.0 -1.0 -1.0 6.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 -1.0 -1.0 0.0 0.0 0.0 2.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 2.0 -1.0 -1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.0 5.0 -1.0 0.0 -1.0 0.0 0.0 0.0 -1.0 -1.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.0 -1.0 8.0 -1.0 -1.0 -1.0 -1.0 0.0 -1.0 -1.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.0 2.0 -1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.0 -1.0 -1.0 4.0 -1.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.0 0.0 -1.0 3.0 0.0 -1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.0 -1.0 0.0 0.0 0.0 0.0 0.0 2.0 0.0
0.0 0.0 0.0 0.0 0.0 -1.0 0.0 0.0 0.0 -1.0 -1.0 0.0 0.0 0.0 0.0 0.0 0.0 3.0

Normalized Laplacian Matrix :
1.0 0.0 0.0 0.0 0.0 0.0 -0.40825 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 -0.28868 0.0 0.0 -0.28868 -0.2357 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 -0.28868 1.0 -0.22361 0.0 0.0 -0.20412 -0.35355 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 -0.22361 1.0 -0.31623 -0.22361 -0.18257 -0.31623 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 -0.31623 1.0 0.0 -0.28868 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 -0.28868 0.0 -0.22361 0.0 1.0 -0.20412 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -0.28868
-0.40825 -0.2357 -0.20412 -0.18257 -0.28868 -0.20412 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0
0.0 0.0 -0.35355 -0.31623 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 -0.31623 -0.25 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -0.31623 1.0 -0.15811 0.0 -0.22361 0.0 0.0 0.0 -0.31623 -0.2582
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -0.25 -0.15811 1.0 -0.25 -0.17678 -0.20412 -0.35355 0.0 -0.25
-0.20412
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -0.25 1.0 -0.35355 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -0.22361 -0.17678 -0.35355 1.0 -0.28868 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -0.20412 0.0 -0.28868 1.0 0.0 -0.57735 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -0.35355 0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -0.57735 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -0.31623 -0.25 0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 0.0 -0.28868 0.0 0.0 0.0 -0.2582 -0.20412 0.0 0.0 0.0 0.0 0.0 0.0 1.0

Eigen-value #1 : 9.03911132504
```

```
( -0.00115014985228 0.00452454501336 -0.0041208728232 0.00747069649449 -0.00237485648763
-0.0324495409106 0.00924618270292 -0.000475887298355 0.117453404358 0.110693416716 -
0.937461005496 0.117100757269 0.113175738832 0.139361684245 0.11661251693 -0.0173354589345
0.117453404358 0.142275424884 )

Eigen-value #2 : 7.12988326346
( 0.146594901887 0.117326259143 0.21094690178 0.201431725722 0.135905219292 0.203118979808
-0.898609635585 -0.0803875266401 -0.00539739194852 0.0464057252201 -0.0187177345971
0.00626855373809 -0.01343921431 0.00810662758181 0.0030535221949 -0.00132247666609 -
0.00539739194852 -0.055887044671 )

Eigen-value #3 : 6.50577218305
( -0.00814385007908 -0.209036657536 0.283620057479 -0.621470465069 0.127976350871 0.404376658519
0.0448381832283 0.0749816887904 -0.0873459785366 0.427228202601 -0.0336671222095 0.0534890654502
-0.207342420993 0.0725027250498 0.00611487745773 -0.0131684934718 -0.0873459785366 -
0.227606843014 )

Eigen-value #4 : 6.27130968554
( 0.00927153047364 -0.11622127094 0.238267457415 -0.418213908777 0.109354519234 0.190801420262
-0.0488731083854 0.0421291043288 0.159505852626 -0.670266587953 -0.011032305268 -0.0897068004582
0.394197830923 -0.124339657714 0.00209289643867 0.0235880009204 0.159505852626 0.149939174248
)

Eigen-value #5 : 4.65157861968
( 0.0065159741134 0.00849735318763 0.295601410626 -0.105688880846 0.0488322207926 -
0.285841865715 -0.0237935917589 -0.0716224396931 0.08356475812 -0.276994972063 0.0554164460733
0.215295918529 -0.626290500548 0.414360107326 -0.0151760243569 -0.113474239632 0.08356475812
0.307233567727 )

Eigen-value #6 : 4.55665193414
( -0.0444651448711 -0.483724172095 0.616937303464 0.367019086532 -0.205411664774 -0.0220941788878
0.158147043508 -0.384861301163 -0.0263860644847 0.073488327598 -0.00602834479871 -0.0447836586114
0.120524572206 -0.0897665604179 0.00169494932603 0.0252390624891 -0.0263860644847 -
0.0291431905338 )

Eigen-value #7 : 4.26556115792
( -0.0212296845659 -0.289656091857 -0.317639286223 0.294443475947 -0.160565212724 0.61488995192
0.0693268333132 0.0102384392471 0.138790900084 -0.289268760245 -0.025170512058 0.111507432752
-0.227456396405 0.263335423556 0.00770786729778 -0.0806401750945 0.138790900084 -0.237405105029
)

Eigen-value #8 : 2.95047962458
( -0.0112285164459 0.116118450393 0.0661916933804 -0.0209287329356 -0.00102291472774 -
0.0823424566653 0.021900992542 -0.0476211791126 -0.0760295368511 0.0194458245091 0.0528187011338
-0.416026259857 0.342605782149 0.703333483317 -0.0270798528056 -0.36059514514 -0.0760295368511
-0.203510796032 )

Eigen-value #9 : 2.60136341397
( 0.0643221421815 -0.466361420951 -0.197866292503 0.0549232630126 0.079951425491 0.114960692997
-0.103003125198 0.237698247299 -0.256434781023 0.110707871216 0.043502624161 -0.171590082791
0.0596853738296 0.100857734903 -0.0271659910433 -0.0629824148742 -0.256434781023 0.675229514317
)

Eigen-value #10 : 2.0
( 1.31486791762e-17 -2.19808193324e-16 -1.09222974552e-16 2.73365677728e-17 2.0310131079e-
16 -2.5773722994e-17 -1.34944841966e-17 6.03091613009e-17 -0.707106781187 -7.58995368411e-
17 -7.54076888266e-17 -8.3059978223e-19 8.29746202539e-17 2.40128393793e-17 -5.00192804578e-
19 -1.9342622484e-17 0.707106781187 1.2428835786e-17 )

Eigen-value #11 : 1.92864027788
( -0.0659083464412 0.494024282704 0.0795057721037 -0.100223023023 -0.546777323558 0.38856680098
0.0612051451539 -0.290321350823 -0.192629868843 -0.0110183422614 -0.00272767165094 0.0219063211959
0.00429090064411 0.000727691482205 0.00293727476172 -0.000783609649007 -0.192629868843
0.349855216068 )

Eigen-value #12 : 1.75764405531
( 0.17157027189 -0.0747604123205 0.187797164092 -0.0290712424643 -0.656309211798 -0.150687010199
-0.129989196567 0.654928938648 0.0882636622503 0.0185945370298 0.00279668621629 -0.0501912799555
-0.0149608412849 -0.00474747468312 -0.0036912930243 0.00626610167376 0.0882636622503
-0.104073061755 )

Eigen-value #13 : 1.58760225114
( 0.0199544612483 -0.0469760509676 0.0289904629486 0.0170345095512 0.0128740353604 -
0.0836140452357 -0.0117252863498 0.111603355322 -0.345372989518 -0.158727799423 0.0162967560293
0.749633336244 0.292850344307 0.0992692185094 -0.027734332191 -0.168939479583 -0.345372989518
-0.160043506733 )

Eigen-value #14 : 1.0362691059
( -0.0803513676319 0.0088044446254 0.0417071799841 0.0336730484714 0.0379642501425 -
0.02733189233 0.00291427226216 0.0782170924656 -0.229188575538 -0.189412600471 -0.0314635103488
-0.172254814189 -0.134543775742 -0.00562061976622 0.867501681244 0.154969901412 -0.229188575538
-0.126396139051 )

Eigen-value #15 : 0.947464298814
( -0.88576477406 0.108549425113 0.180261274273 0.16254407566 0.110219351267 0.089074569606
-0.0465342734908  0.325694747976  0.00769649680838  0.014259480659  -0.00615864299407  -
0.0047029432519 0.0012086273208 0.00029148363318 -0.117227768072 0.00554829623669 0.00769649680838
0.0473440765073 )

Eigen-value #16 : 0.649139061158
```

(0.0566069151158 0.00756432330576 0.0217975106848 0.0169393011377 0.0272422241713 -0.023875993879 0.0198611553825 0.0286756472918 -0.198762414217 -0.152665427921 -0.115834953555 -0.105363401541 -0.0264963499704 0.28507819888 -0.330144911365 0.812510505788 -0.198762414217 -0.124369915091)

Eigen-value #17 : 0.121529742416
(-0.296055197122 -0.246460620983 -0.270481156077 -0.260075829888 -0.276901650725 -0.178872725814 -0.260075685275 -0.282440982935 0.20686879906 0.186467888326 0.202128997931 0.225226275572 0.220951861957 0.243132063128 0.23009202211 0.276767552492 0.20686879906 0.0728595891829)

Eigen-value #18 : -5.86745193283e-15
(0.0028507561306 0.187422549019 -0.251378305306 0.580590259547 -0.112421663761 -0.390706693913 -0.0496898298449 -0.0651928381797 0.0962311945541 -0.480572421412 0.0167161781781 -0.0687031048986 0.270822926979 -0.0835552708581 -0.00597347692893 0.0105323723255 0.0986973239003 0.239070835928)

# of real eigen-values : 18
# of complex eigen-values : 0
# of real eigen-values : 18

I have used the Power Method (Iterative Power method with deflation techniques) to find the eigenvalues and eigenvectors
`https://en.wikipedia.org/wiki/Power_iteration`

`https://en.wikipedia.org/wiki/Laplacian_matrix#Definition`

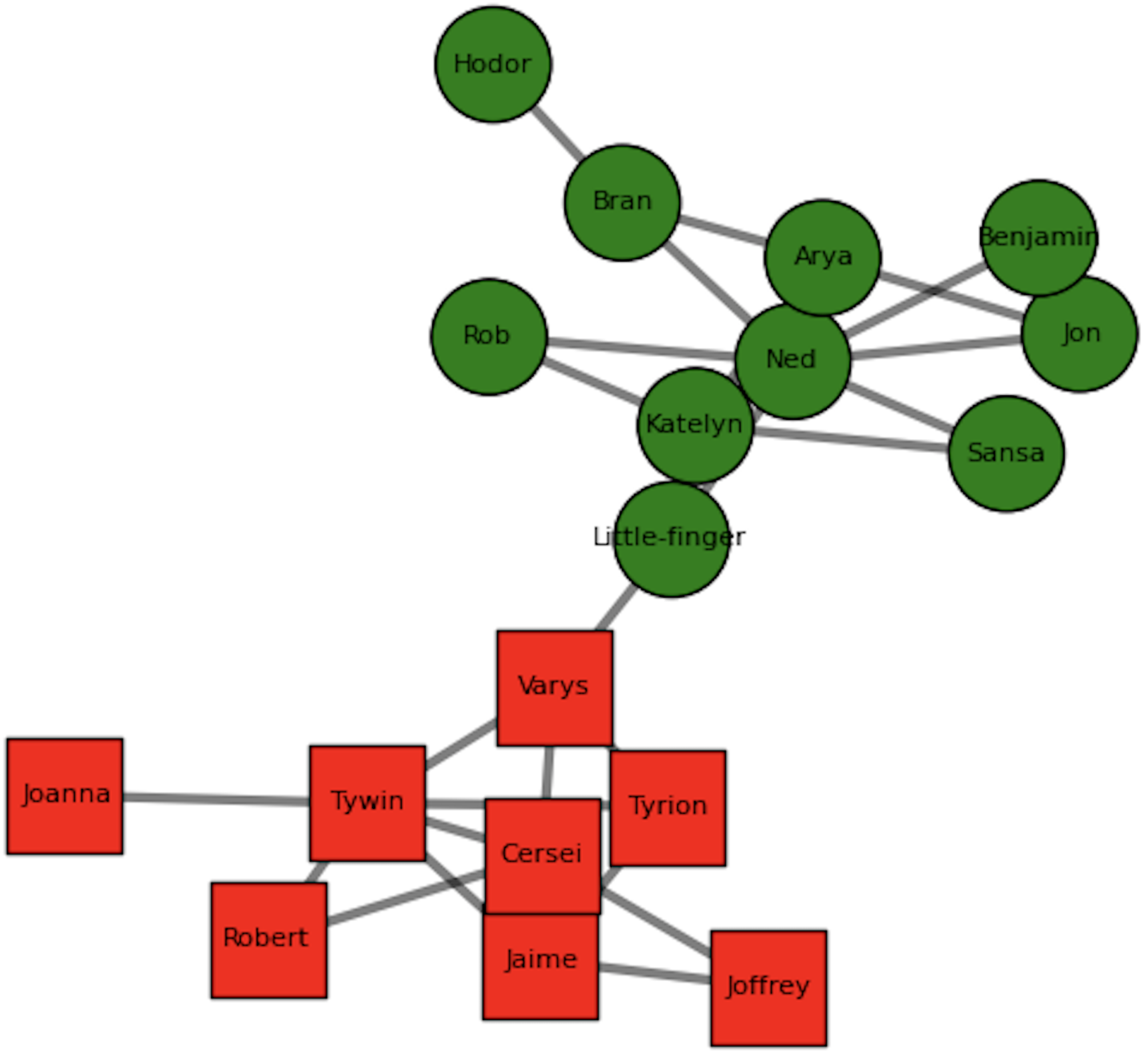
Task - 5
Note down the the smallest two eigenvalues and their corresponding eigenvectors. Ideally, you should get the smallest eigenvalue to be (approximately) zero and components of eigenvector to be all (approximately) one. Tally these values and report how much your answer differs from the ideal values.

Solution:
Smallest Eigen-value : -4.19812828955e-15
Second Smallest Eigen-value : 0.121529742416
Absolute difference between eigen-value and zero : -4.19812828955e-15
Absolute difference (component wise) between the smallest eigen-vector and vector of ones (Both normalized) : 3.67977691904

Task - 6
Take the eigenvector corresponding to the second smallest eigenvalue obtained earlier. Find two clusters in the graph based on positive and negative entries of the eigenvector. Try out a different technique other than this which you think more appropriate to divide these entries into two groups. Give two different colors to the nodes and draw the graph. Save it. What

do you observe? Can you infer something from it? Use networkx.draw for visualization.

Solution:



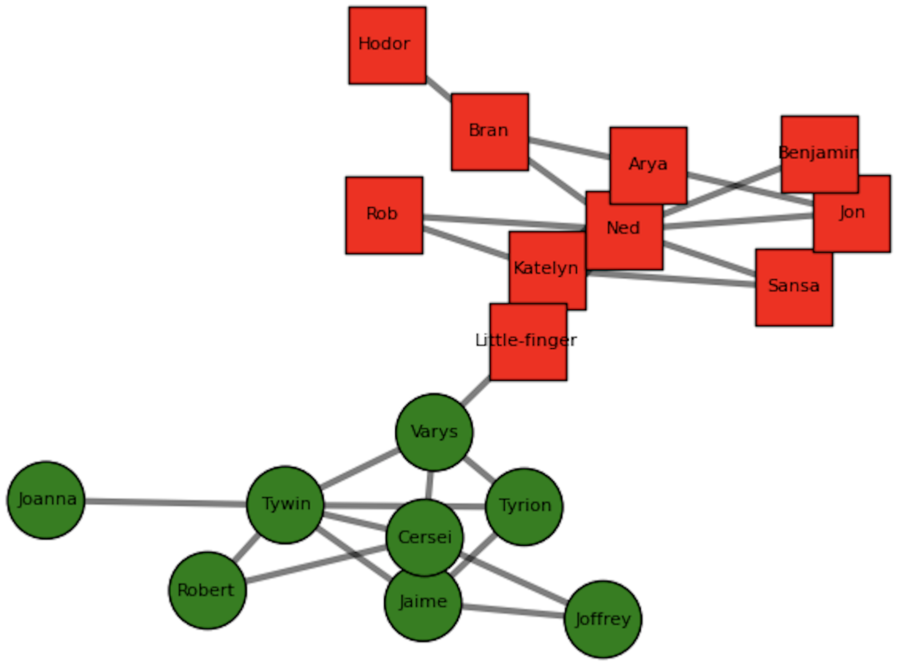
Clustering w.r.t second smallest eigen value :
0.121529742416
(0.296055197122 0.246460620983 0.270481156077 0.260075829888 0.276901650725 0.178872725814
0.260075685275 0.282440982935 -0.20686879906 -0.186467888326 -0.202128997931 -0.225226275572
-0.220951861957 -0.243132063128 -0.23009202211 -0.276767552492 -0.20686879906 -0.0728595891829
)
House 1: Sansa, Katelyn, Ned, Jon, Arya, Bran, Benjamin, Hodor, Rob, Little-finger,
House 2: Joanna, Tyrion, Jaime, Cersei, Robert, Varys, Tywin, Joffrey,

Task - 7
The House of Algebrician wants to connect with that house which contains more information. If the information is measured in terms of the variability in the data, which house would they want to make friends with?

Solution: Using maximum of variance as a measure for the two clusters.
Clustering w.r.t second smallest eigen value :
House 1: Sansa, Katelyn, Ned, Jon, Arya, Bran, Benjamin, Hodor, Rob, Little-finger,
House 2: Joanna, Tyrion, Jaime, Cersei, Robert, Varys, Tywin, Joffrey,
Result: (Based on the variability measure) House of Algebrician should join House 1

Bonus - 1
Use numpy to find the eigenvectors and cluster them into two groups. Check to see if you find any difference.

Solution:
Clustering w.r.t second smallest eigen value (USING NUMPY):
0.12152974241615139
(-0.11745340435795648 -0.0053973919485162765 0.0873459785365985 0.15950585262601577 -0.08356475812001754 -0.02638606448472425 -0.13879090008393757 0.2357022603955147 0.20686879906048292 0.0760295368510997 -0.2564347810234897 -0.1987624142172698 0.007696496808376677 -0.22918857553801228 -0.34537298951828754 0.08826366225032922 -0.19262986884341643 -0.7071067811865483)
House 1: Sansa, Katelyn, Ned, Jon, Arya, Bran, Benjamin, Hodor, Rob, Little-finger,
House 2: Joanna, Tyrion, Jaime, Cersei, Robert, Varys, Tywin, Joffrey,

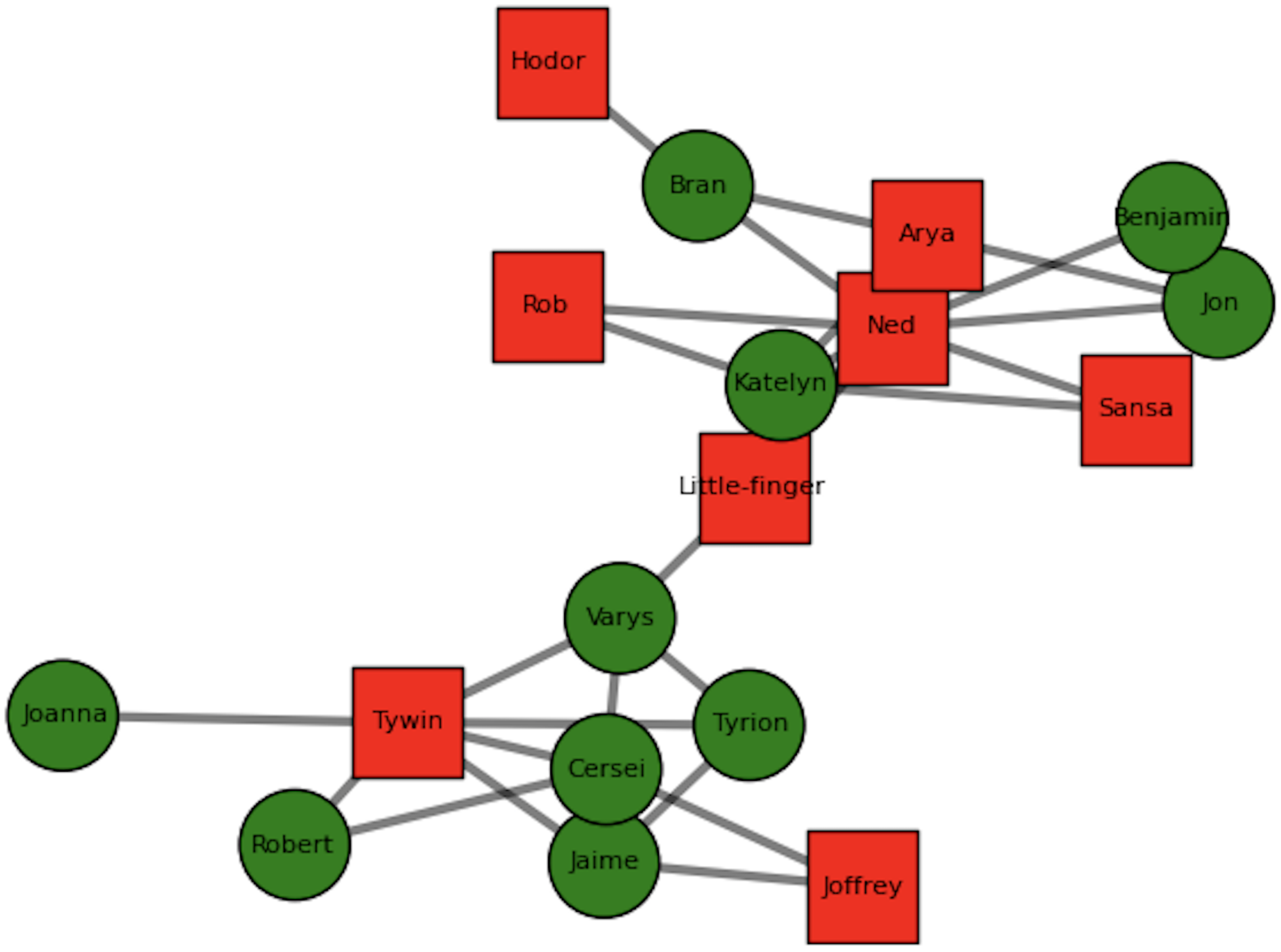


Bonus - 2
What would you do if you are asked to find three or more clusters? How can you extend this idea to find more than two clusters in a graph?

Solution:
We can use a hierarchical clustering, i.e. after finding the two clusters (using the above procedure), we can recurse on the smaller clusters to find more clusters.

Bonus - 3
What do you observe if you take the eigenvector corresponding to the second largest eigenvalue instead of the second smallest for the clustering process? As before, color and draw the network.

Solution:



Clustering w.r.t second largest eigen value :
7.12988326346
(0.146594901887 0.117326259143 0.21094690178 0.201431725722 0.135905219292 0.203118979808 -0.898609635585 -0.0803875266401 -0.00539739194852 0.0464057252201 -0.0187177345971 0.00626855373809 -0.01343921431 0.00810662758181 0.0030535221949 -0.00132247666609 -0.00539739194852 -0.055887044671)
House 1: Joanna, Tyrion, Jaime, Cersei, Robert, Varys, Katelyn, Jon, Bran, Benjamin,
House 2: Tywin, Joffrey, Sansa, Ned, Arya, Hodor, Rob, Little-finger,

Bonus - 4
How do you relate the central edge in the graph to the clustering obtained?

Solution: This acts as a bridge of the network, i.e. an edge if removed from the graph will disconnect the graph into two components.

Problem 2

Task - 1
Find the mean and covariance matrix of the given data. Don't use any library function to calculate these.

Solution: I have used the following references to compute the given matrices,
`https://en.wikipedia.org/wiki/Covariance_matrix`
`https://www.itl.nist.gov/div898/handbook/pmc/section5/pmc541.htm`

Task - 2
Obtain the eigenspaces of the given data. How many eigenvalues are repeating? What can you say about the eigenvectors obtained for different eigenspaces? Do they turn out to be orthogonal?

Solution: To compute the number of **distinct** eigen-values, I have used different error tolerance (for comparison) and the result is as follows:
Within 0.01 : 648
Within 0.001 : 683
Within 0.0001 : 733
Within 0.00001 : 772

Yes, The eigen-vectors corresponding to different eigenspaces (different underlying eigen-value) are orthogonal to each other.

Task - 3
Use the process of Gram-Scmidt orthogonalization to get the orthogonal eigenvectors of different eigenspaces (eigenvalues with multiplicity more than one). You will be given a set

of vectors as input and the output should be a set of orthogonal vectors.

Solution:
Test 1:
Input:

```
1 0 0 0 0
0.71 0.71 0 0 0
0.58 0.58 0.58 0 0
0.50 0.50 0.50 0.50 0
0.45 0.45 0.45 0.45 0.45
```

Output:

```
1.0 0.0 0.0 0.0 0.0
0.0 0.71 0.0 0.0 0.0
0.0 0.0 0.58 0.0 0.0
0.0 0.0 0.0 0.5 0.0
0.0 0.0 0.0 0.0 0.45
```

Test 2:
Input:

```
1 0 0 0 0
2 0 0 0 0
3 0 0 0 0
0.71 0.71 0 0 0
0.58 0.58 0.58 0 0
0.50 0.50 0.50 0.50 0
0.45 0.45 0.45 0.45 0.45
0 0 0 0 0
```

Output:

```
3.0 0.0 0.0 0.0 0.0
0.0 0.71 0.0 0.0 0.0
0.0 0.0 0.58 0.0 0.0
0.0 0.0 0.0 0.5 0.0
0.0 0.0 0.0 0.0 0.45
0.0 0.0 0.0 0.0 0.0
```

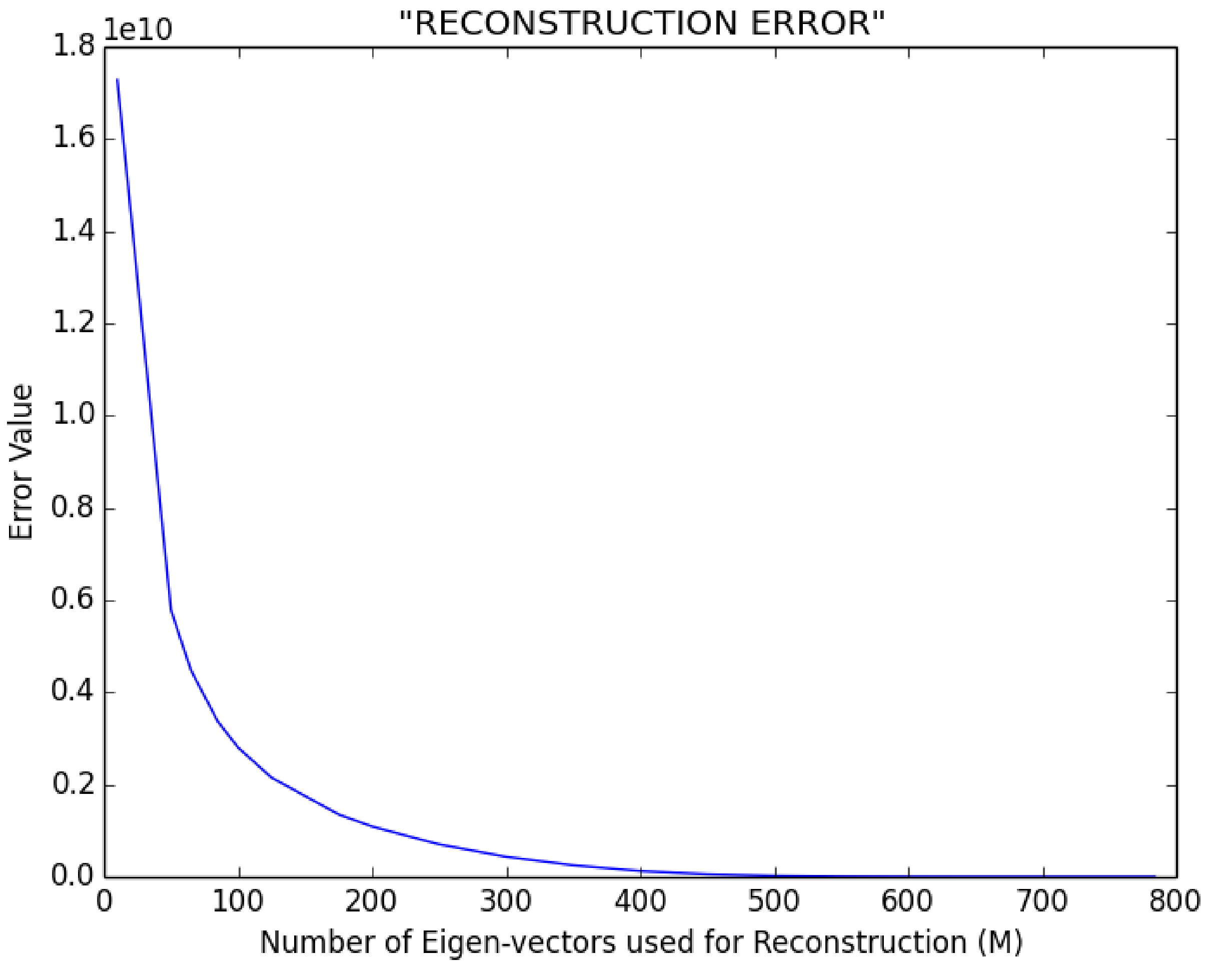
I have used the following references for the procedure,
`https://en.wikipedia.org/wiki/Gram-Schmidt_process` and class notes

Task - 4
Pick M eigenvectors corresponding to the top-M eigenvalues obtained from the covariance matrix. Use these vectors to convert the original data into M- dimensional space (the original data is D-dimensional and M$\leq$D).

Solution: I have used the following references for dimensionality reduction using Principal Component Analysis (PCA)
`https://en.wikipedia.org/wiki/Principal_component_analysis`

Task - 5
Calculate and plot reconstruction error for different values of M. Set the rest of the (D - M) dimensions to be zero.

Solution: I have used the following references for reconstruction error in PCA
`https://en.wikipedia.org/wiki/Principal_component_analysis`
`https://en.wikipedia.org/wiki/Matrix_norm`
`http://www.cs.princeton.edu/courses/archive/spr08/cos424/scribe_notes/0424.pdf`
Here the error is calculated by the frobenius norm between the original data matrix and the matrix obtained after reconstruction of the dimensionally reduced data matrix for different values of M.



Task - 6
Find the mean of each column of the given data and subtract it from original data. This is called Data Centering in statistics. Divide your dataset into training and validation sets using cross validation technique. Use K-Nearest Neighbor to classify the validation points. Use euclidean distance or cosine similarity for calculating the nearest neighbour. Report the accuracy for different values of K and M. Plot the results. What do you observe as you vary K and M? Finalize your model fixing K and M which give best accuracy.

Solution: I have fixed K = 3 and M = 250 in my model. For the training and testing datasets given, my results are:

Class-wise Results-

Class 0 Correct : 108 Incorrect : 0 Total : 108 Accuracy Percentage : 100.0 Error Percentage : 0.0

Class 1 Correct : 98 Incorrect : 0 Total : 98 Accuracy Percentage : 100.0 Error Percentage : 0.0

Class 2 Correct : 80 Incorrect : 8 Total : 88 Accuracy Percentage : 90.9090909091 Error Percentage : 9.09090909091

Class 3 Correct : 101 Incorrect : 4 Total : 105 Accuracy Percentage : 96.1904761905 Error Percentage : 3.80952380952

Class 4 Correct : 90 Incorrect : 9 Total : 99 Accuracy Percentage : 90.9090909091 Error Percentage : 9.09090909091

Class 5 Correct : 85 Incorrect : 12 Total : 97 Accuracy Percentage : 87.6288659794 Error Percentage : 12.3711340206

Class 6 Correct : 99 Incorrect : 3 Total : 102 Accuracy Percentage : 97.0588235294 Error Percentage : 2.94117647059

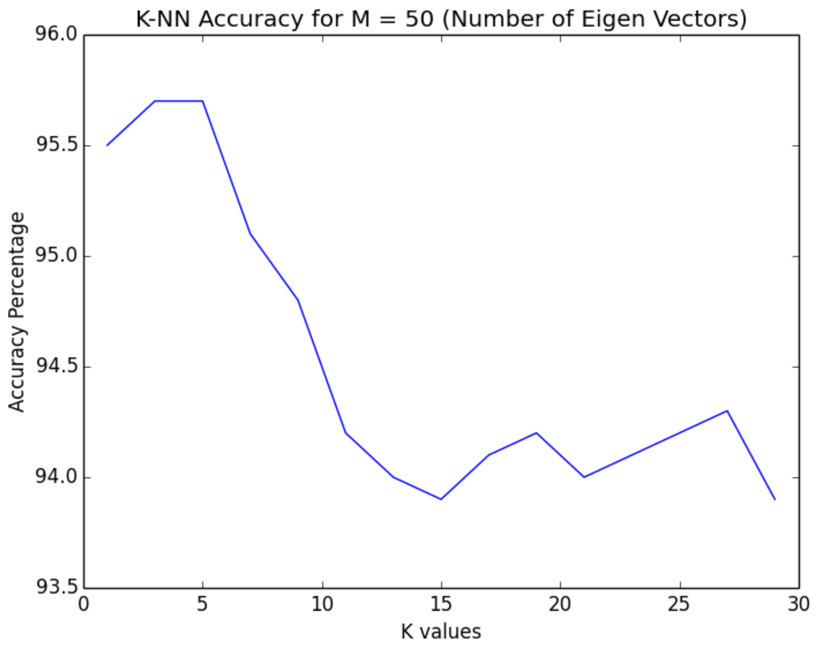
Class 7 Correct : 90 Incorrect : 5 Total : 95 Accuracy Percentage : 94.7368421053 Error Percentage : 5.26315789474

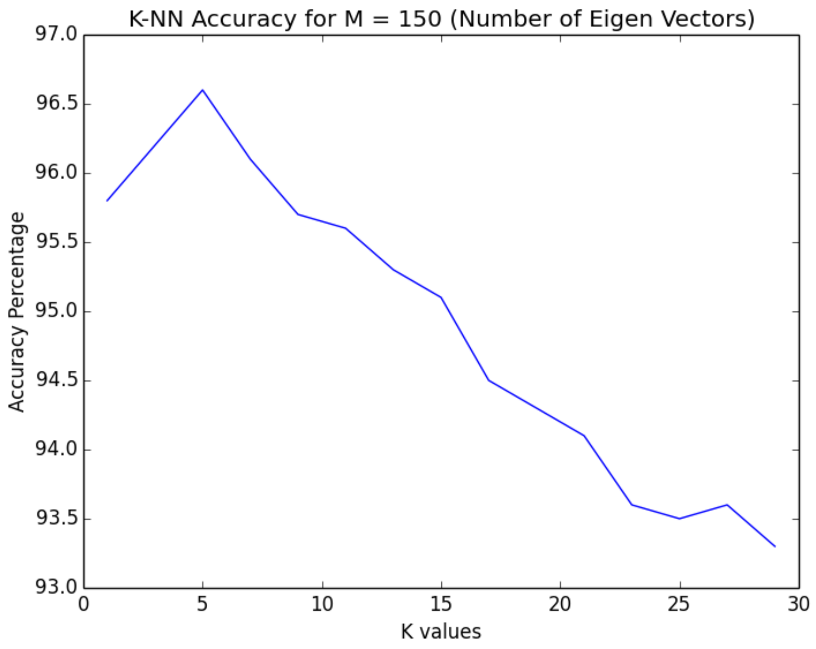
Class 8 Correct : 95 Incorrect : 16 Total : 111 Accuracy Percentage : 85.5855855856 Error Percentage : 14.4144144144

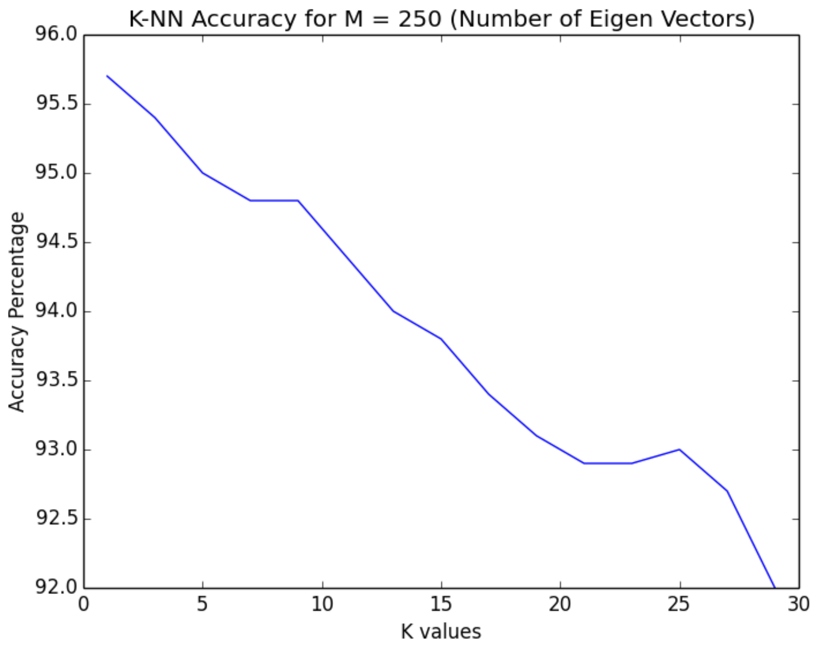
Class 9 Correct : 85 Incorrect : 12 Total : 97 Accuracy Percentage : 87.6288659794 Error Percentage : 12.3711340206

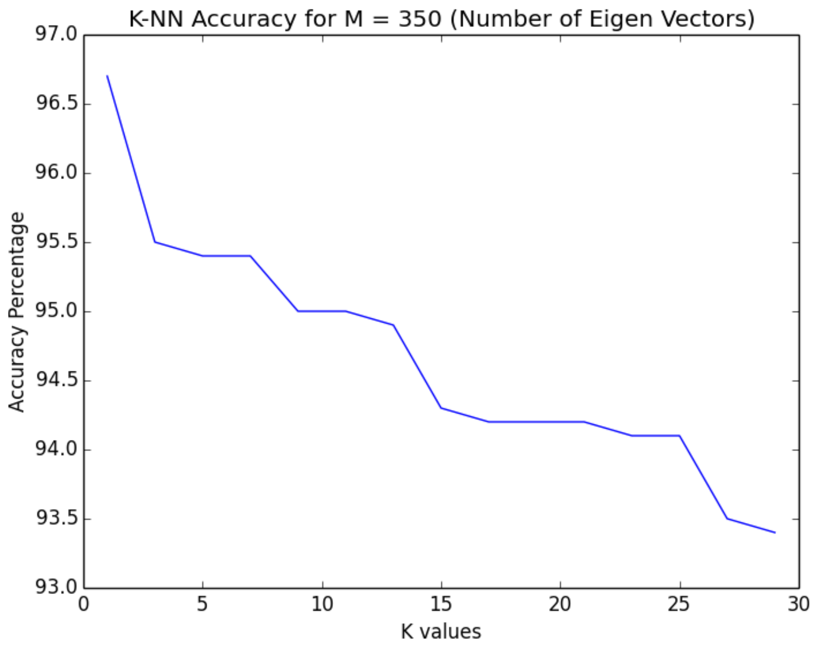
Aggregate Results-
Correct : 931
Incorrect : 69
Total : 1000
Accuracy Percentage : 93.1
Error Percentage : 6.9

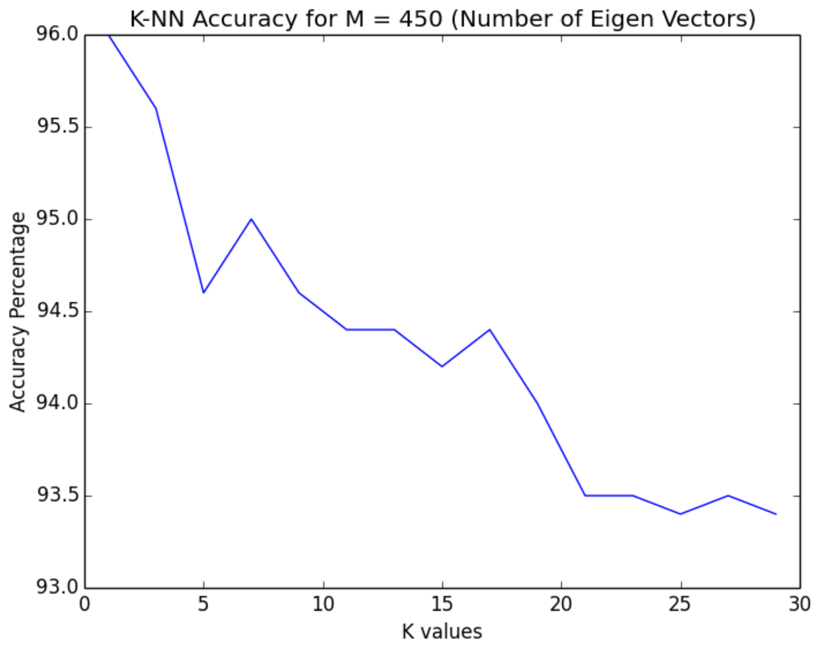
Using euclidean distance as the measure, I have observed the following results,

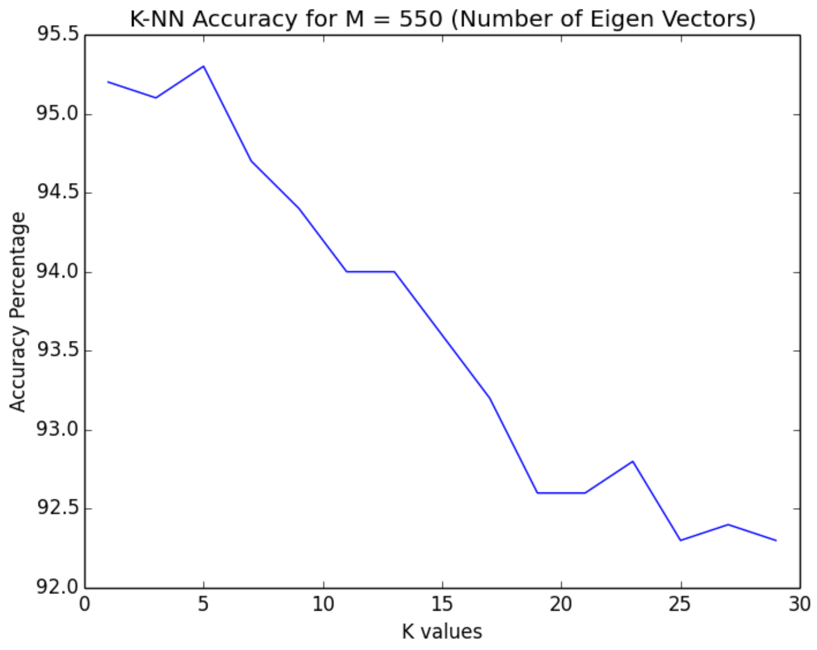

K-NN Accuracy for M = 50 (Number of Eigen Vectors)
Accuracy Percentage
K values
96.0
95.5
95.0
94.5
94.0
93.5
0
5
10
15
20
25
30

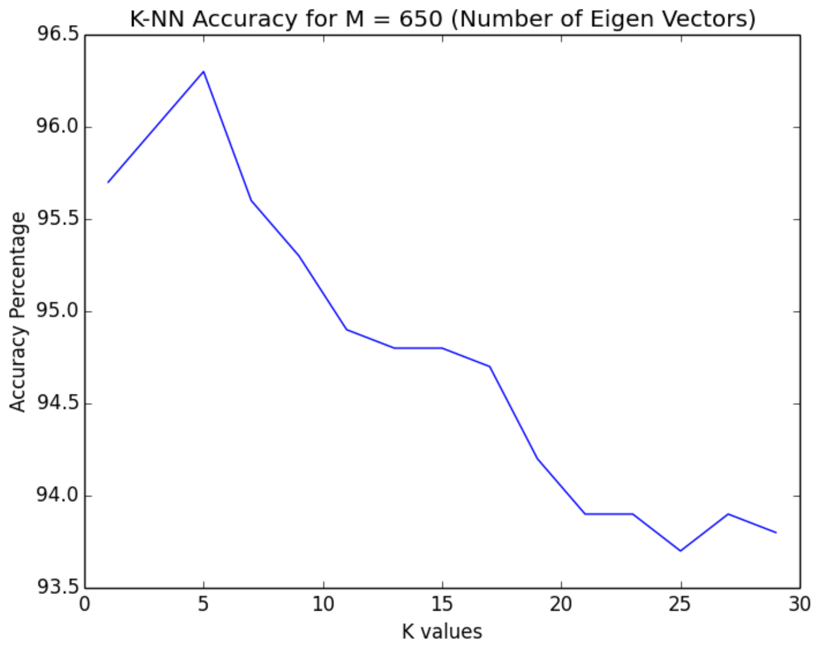

K-NN Accuracy for M = 150 (Number of Eigen Vectors)
Accuracy Percentage
K values
97.0
96.5
96.0
95.5
95.0
94.5
94.0
93.5
93.0
0
5
10
15
20
25
30

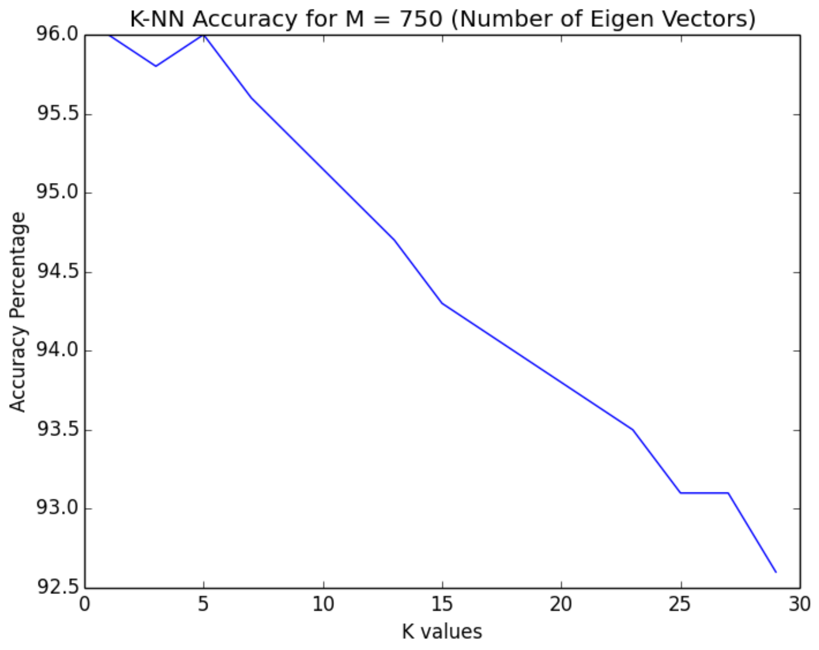

K-NN Accuracy for M = 250 (Number of Eigen Vectors)
Accuracy Percentage
K values
96.0
95.5
95.0
94.5
94.0
93.5
93.0
92.5
92.0
0
5
10
15
20
25
30


K-NN Accuracy for M = 350 (Number of Eigen Vectors)
Accuracy Percentage
K values
97.0
96.5
96.0
95.5
95.0
94.5
94.0
93.5
93.0
0
5
10
15
20
25
30


K-NN Accuracy for M = 450 (Number of Eigen Vectors)
Accuracy Percentage
K values
96.0
95.5
95.0
94.5
94.0
93.5
93.0
0
5
10
15
20
25
30

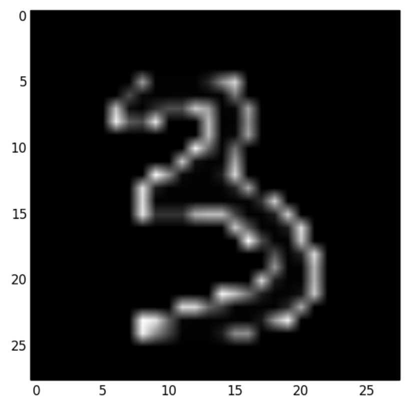

K-NN Accuracy for M = 550 (Number of Eigen Vectors)
Accuracy Percentage
K values
95.5
95.0
94.5
94.0
93.5
93.0
92.5
92.0
0
5
10
15
20
25
30

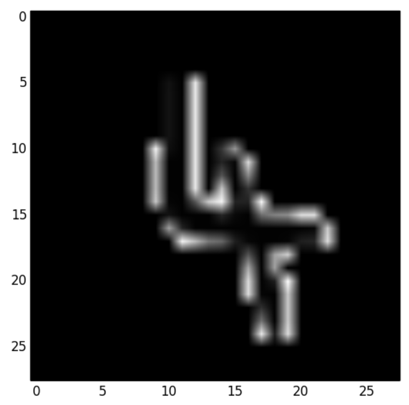

K-NN Accuracy for M = 650 (Number of Eigen Vectors)
Accuracy Percentage
K values
96.5
96.0
95.5
95.0
94.5
94.0
93.5
0
5
10
15
20
25
30

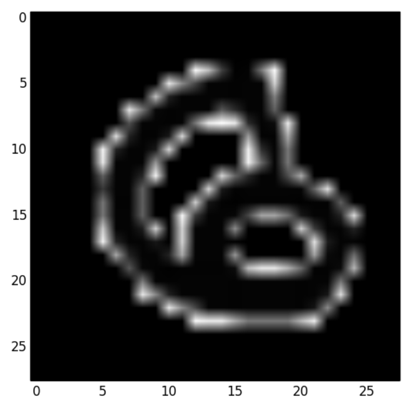

K-NN Accuracy for M = 750 (Number of Eigen Vectors)
Accuracy Percentage
K values
96.0
95.5
95.0
94.5
94.0
93.5
93.0
92.5
0
5
10
15
20
25
30

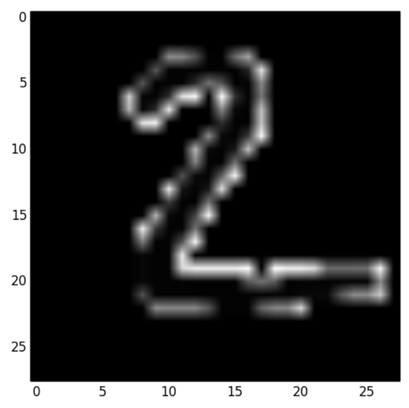
Bonus - 1
Project the vectors into 2D space and plot them. Do you see any pattern in the plot?

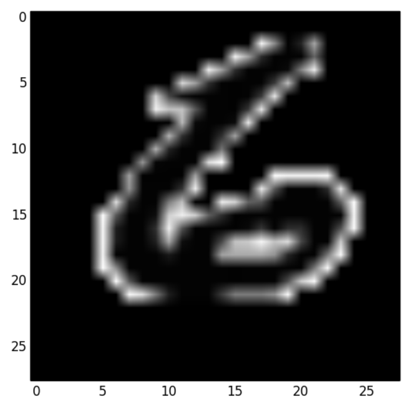
Solution: Some of plots of data when projected to 2D are below:






0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25

Bonus - 2
Use K-NN classifier available in sklearn and compare your results.

Solution: Using Sklearn we get the following accuracies, Using Sklearn, I obtained the below data,
M = 50
K=1, accuracy=95.50%
K=3, accuracy=95.70%
K=5, accuracy=95.70%
K=7, accuracy=95.10%
K=9, accuracy=94.80%
K=11, accuracy=94.20%
K=13, accuracy=94.00%
K=15, accuracy=93.90%
K=17, accuracy=94.10%
K=19, accuracy=94.20%
K=21, accuracy=94.00%
K=23, accuracy=94.10%
K=25, accuracy=94.20%
K=27, accuracy=94.30%
K=29, accuracy=93.90%
K=3 achieved highest accuracy of 95.70% on validation data

M = 150
K=1, accuracy=95.80%
K=3, accuracy=96.20%
K=5, accuracy=96.60%
K=7, accuracy=96.10%
K=9, accuracy=95.70%
K=11, accuracy=95.60%
K=13, accuracy=95.30%
K=15, accuracy=95.10%
K=17, accuracy=94.50%
K=19, accuracy=94.30%
K=21, accuracy=94.10%
K=23, accuracy=93.60%
K=25, accuracy=93.50%
K=27, accuracy=93.60%
K=29, accuracy=93.30%
K=5 achieved highest accuracy of 96.60% on validation data

M = 250
K=1, accuracy=95.70%
K=3, accuracy=95.40%
K=5, accuracy=95.00%

```
K=7, accuracy=94.80%
K=9, accuracy=94.80%
K=11, accuracy=94.40%
K=13, accuracy=94.00%
K=15, accuracy=93.80%
K=17, accuracy=93.40%
K=19, accuracy=93.10%
K=21, accuracy=92.90%
K=23, accuracy=92.90%
K=25, accuracy=93.00%
K=27, accuracy=92.70%
K=29, accuracy=92.00%
K=1 achieved highest accuracy of 95.70% on validation data

M = 350
K=1, accuracy=96.70%
K=3, accuracy=95.50%
K=5, accuracy=95.40%
K=7, accuracy=95.40%
K=9, accuracy=95.00%
K=11, accuracy=95.00%
K=13, accuracy=94.90%
K=15, accuracy=94.30%
K=17, accuracy=94.20%
K=19, accuracy=94.20%
K=21, accuracy=94.20%
K=23, accuracy=94.10%
K=25, accuracy=94.10%
K=27, accuracy=93.50%
K=29, accuracy=93.40%
K=1 achieved highest accuracy of 96.70% on validation data

M = 450
K=1, accuracy=96.00%
K=3, accuracy=95.60%
K=5, accuracy=94.60%
K=7, accuracy=95.00%
K=9, accuracy=94.60%
K=11, accuracy=94.40%
K=13, accuracy=94.40%
K=15, accuracy=94.20%
K=17, accuracy=94.40%
K=19, accuracy=94.00%
K=21, accuracy=93.50%
K=23, accuracy=93.50%
```

```
K=25, accuracy=93.40%
K=27, accuracy=93.50%
K=29, accuracy=93.40%
K=1 achieved highest accuracy of 96.00% on validation data

M = 550
K=1, accuracy=95.20%
K=3, accuracy=95.10%
K=5, accuracy=95.30%
K=7, accuracy=94.70%
K=9, accuracy=94.40%
K=11, accuracy=94.00%
K=13, accuracy=94.00%
K=15, accuracy=93.60%
K=17, accuracy=93.20%
K=19, accuracy=92.60%
K=21, accuracy=92.60%
K=23, accuracy=92.80%
K=25, accuracy=92.30%
K=27, accuracy=92.40%
K=29, accuracy=92.30%
K=5 achieved highest accuracy of 95.30% on validation data

M = 650
K=1, accuracy=95.70%
K=3, accuracy=96.00%
K=5, accuracy=96.30%
K=7, accuracy=95.60%
K=9, accuracy=95.30%
K=11, accuracy=94.90%
K=13, accuracy=94.80%
K=15, accuracy=94.80%
K=17, accuracy=94.70%
K=19, accuracy=94.20%
K=21, accuracy=93.90%
K=23, accuracy=93.90%
K=25, accuracy=93.70%
K=27, accuracy=93.90%
K=29, accuracy=93.80%
K=5 achieved highest accuracy of 96.30% on validation data

M = 750
K=1, accuracy=96.0%
K=3, accuracy=95.80%
K=5, accuracy=96.00%
```

```
K=7, accuracy=95.60%
K=9, accuracy=95.30%
K=11, accuracy=95.00%
K=13, accuracy=94.70%
K=15, accuracy=94.30%
K=17, accuracy=94.10%
K=19, accuracy=93.90%
K=21, accuracy=93.70%
K=23, accuracy=93.50%
K=25, accuracy=93.10%
K=27, accuracy=93.10%
K=29, accuracy=92.60%
K=1 achieved highest accuracy of 96.00% on validation data
```