**Reference Pages**

**KMeans Results**

============ BestK Array ============

[0.32275869091473375, 0.34764724923757118, 0.33866421104329725, 0.34638135390827729, 0.32575111171447835, 0.32848141292630906, 0.31315356058413479, 0.31832040622321156, 0.31648028791959371, 0.31904025245115764, 0.31960587720249939, 0.32054106620930595, 0.321129336027901, 0.31648087809478054, 0.31092805158523074, 0.31300520439377105, 0.3122994842662204, 0.30861379507265418, 0.30902109518610843, 0.30576496405350134, 0.30613192387169058, 0.30371027190526634, 0.29971218061186927, 0.30306239057784162, 0.30119913785882402, 0.3007101425460032, 0.30140303091032183, 0.30064108192011241, 0.30300110136761171, 0.30361844787607317, 0.3017514430777406, 0.30467726040380522, 0.30425299384868582, 0.30507847714856962, 0.30452802039669508, 0.30763357194013913, 0.30575129949681312, 0.30544633993126685, 0.30551563708807306, 0.30724674052460677, 0.30669594822270829, 0.30382837312105859, 0.30237644352314785, 0.30264603458452255, 0.30149185807483242, 0.30061814961556588, 0.30423920901840312, 0.30348836361071163, 0.30297605031088742, 0.30255252507053632, 0.30250188760544006, 0.30098832629245875, 0.30123208538526824, 0.30291883499670541, 0.30372504622638019, 0.30157338190586863, 0.30341007742686232, 0.30132208383240822, 0.30358136844704081, 0.30278541817900473, 0.30077679702321108, 0.30359355645257358, 0.30135321131581233, 0.3030482732066529, 0.30317255332048582, 0.3017559443130664, 0.30377457109315198, 0.30317976926709839, 0.3049207548742362, 0.30151074538680472, 0.3068646961685203, 0.30436423133928464, 0.30238066448126755, 0.3022251313436829, 0.30447080337784699, 0.30337073762178413, 0.30278897309021235, 0.30322161475723569, 0.30474378455332901, 0.30411644240281588, 0.30314404476675494, 0.30319297065094197, 0.3043519478599544, 0.30558587641600826, 0.30408363749237477, 0.30476981997861935, 0.30362591393998678, 0.30403345367546086, 0.30428300395103708, 0.30363918242214138, 0.30438168773539143, 0.30558286567039195, 0.30602840634334699, 0.30463085651286914, 0.30391930447846044, 0.30507362796999493, 0.30599399529457877, 0.30728476396143589, 0.30349791946531096]

============ BestK ============

0.347647249238

============ BestK Index ============

1

**Spectral Clustering Results**

============ Best Spectral Array ============

[0.078790698768848838, -0.068653104840236498, -0.19039401943456982, -0.21783814438250965, -0.26770676796197179, -0.3549583782173325]

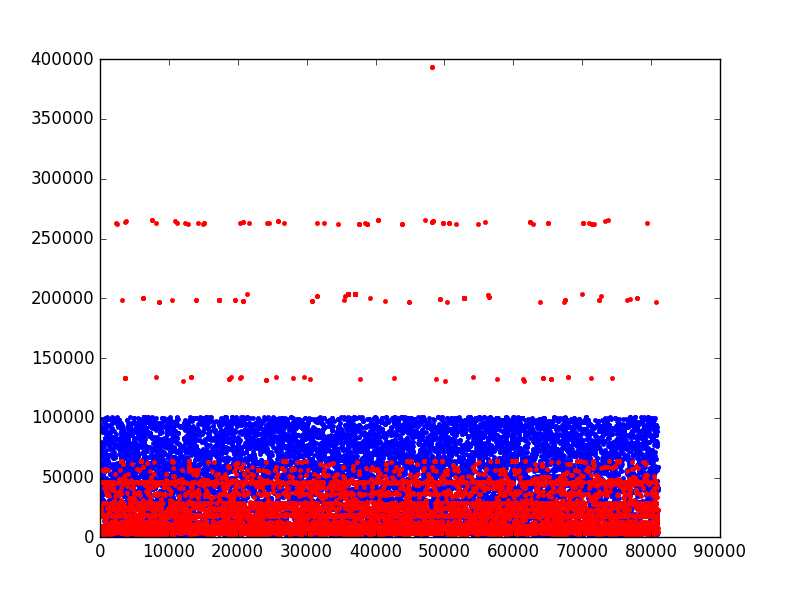
============ Best Spectral Score ============

0.078790698768848838

============ Best Spectral Score Index ============

0

**Generated Graph**



**KMeans.py**

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** matplotlib.colors **import** ListedColormap  
**from** sklearn.cluster **import** KMeans  
**from** sklearn **import** metrics  
**from** sklearn.metrics **import** pairwise\_distances  
**from** numpy **import** genfromtxt  
**from** sklearn.model\_selection **import** KFold  
**import** time  
**import** os  
  
*# Start Timer*start\_time = time.time()  
  
*# How many times to run KMeans.py*run\_times = 1  
  
**for** i **in** range(run\_times):  
 *# read digits data & split it into X (training input) and y (target output)* X = genfromtxt(**'Dataset/IP-Port.csv'**, delimiter=**' '**)  
  
 plt.plot(X[:, 0], X[:, 1], **'.'**)  
 plt.plot(X[:, 0], X[:, 2], **'r.'**)  
 *# plt.show()* bestk = []  
 kc = 0  
 result\_num = len(os.listdir(**'KMeans\_Results'**))  
 **with** open(**'KMeans\_Results/kmeans\_results'** + str(result\_num) + **'.txt'**, **'w'**) **as** file:  
 **for** clusters **in** range(2, 101, 1):  
 kf = KFold(n\_splits=10)  
 *# clusters = 85* kscore = []  
 k = 0  
  
 print(**"============ KScore ============"**)  
 file.writelines(**"============ KScore ============"** + **"\n"**)  
 **for** train, test **in** kf.split(X):  
 *#print("%s %s" % (train, test))* X\_train, X\_test = X[train], X[test]  
  
 *#time.sleep(100)  
  
 # we create an instance of Neighbors Classifier and fit the data.* clf = KMeans(n\_clusters=clusters)  
 clf.fit(X\_train)  
  
 labels = clf.labels\_  
 kscore.append(metrics.silhouette\_score(X\_train, labels, metric=**'euclidean'**))  
  
 print(kscore[k])  
 file.writelines(str(kscore[k]) + **"\n"**)  
 k=k+1  
  
 print(**"============ Clusters ============"**)  
 file.writelines(**"============ Clusters ============"** + **"\n"**)  
 print (clusters)  
 file.writelines(str(clusters) + **"\n"**)  
 bestk.append(sum(kscore)/len(kscore))  
 print(**"============ BestK[KC] ============"**)  
 print(bestk[kc])  
 file.writelines(**"============ BestK[KC] ============"** + **"\n"**)  
 file.writelines(str(bestk[kc]) + **"\n"**)  
 kc+=1  
  
 *# to do here: given this array of E\_outs in CV, find the max, its  
 # corresponding index, and its corresponding value of clusters* print(**"============ BestK Array ============"**)  
 file.writelines(**"============ BestK Array ============"** + **"\n"**)  
 print(bestk)  
 file.writelines(str(bestk) + **"\n"**)  
  
 print(**"============ BestK ============"**)  
 file.writelines(**"============ BestK ============"** + **"\n"**)  
 print(max(bestk))  
 file.writelines(str(max(bestk)) + **"\n"**)  
 print(**"============ BestK Index ============"**)  
 file.writelines(**"============ BestK Index ============"** + **"\n"**)  
 print(bestk.index(max(bestk)))  
 file.writelines(str(bestk.index(max(bestk))) + **"\n"**)  
  
 *# Calculates Running Time* run\_time = time.time() - start\_time  
 minutes, seconds = divmod(run\_time, 60)  
 hours, minutes = divmod(minutes, 60)  
 print(**"============ Running Time ============"**)  
 file.writelines(**"============ Running Time ============"** + **"\n"**)  
 print(**"Seconds Time Format --- %s seconds ---"** % (time.time() - start\_time))  
 print(**"Normal Time Format --- %d:%02d:%02d ---"** % (hours, minutes, seconds))  
 file.writelines(str(**"Seconds Time Format --- %s seconds ---"** % (time.time() - start\_time)) + **"\n"**)  
 file.writelines(str(**"Normal Time Format --- %d:%02d:%02d ---"** % (hours, minutes, seconds)))

**Spectral Clustering.py**

**import** matplotlib.pyplot **as** plt  
**from** sklearn **import** metrics  
**from** numpy **import** genfromtxt  
**from** sklearn.model\_selection **import** KFold  
**from** sklearn.cluster **import** SpectralClustering  
**import** os  
**import** time  
  
*# Start Timer*start\_time = time.time()  
  
*# How many times to run KMeans.py*run\_times = 1  
  
**for** i **in** range(run\_times):  
 *# read digits data & split it into X (training input) and y (target output)* X = genfromtxt(**'Dataset/IP-Port.csv'**, delimiter=**' '**)  
  
 plt.plot(X[:, 0], X[:, 1], **'.'**)  
 plt.plot(X[:, 0], X[:, 2], **'r.'**)  
 *# plt.show()* bestk = []  
 kc = 0  
 result\_num = len(os.listdir(**'KMeans\_Results'**))  
 **with** open(**'Spectral\_Clustering\_Results/spectral\_clustering\_results'** + str(result\_num) + **'.txt'**, **'w'**) **as** file:  
 **for** clusters **in** range(2, 101, 1):  
 kf = KFold(n\_splits=10)  
 *# clusters = 85* kscore = []  
 k = 0  
  
 print(**"============ Spectral Score ============"**)  
 file.writelines(**"============ Spectral Score ============"** + **"\n"**)  
 **for** train, test **in** kf.split(X):  
 *#print("%s %s" % (train, test))* X\_train, X\_test = X[train], X[test]  
  
 *#time.sleep(100)  
  
 # we create an instance of Neighbors Classifier and fit the data.* clf = SpectralClustering(n\_clusters=clusters)  
 clf.fit(X\_train)  
  
 labels = clf.labels\_  
 kscore.append(metrics.silhouette\_score(X\_train, labels, metric=**'euclidean'**))  
  
 print(kscore[k])  
 file.writelines(str(kscore[k]) + **"\n"**)  
 k=k+1  
  
 print(**"============ Clusters ============"**)  
 file.writelines(**"============ Clusters ============"** + **"\n"**)  
 print (clusters)  
 file.writelines(str(clusters) + **"\n"**)  
 bestk.append(sum(kscore)/len(kscore))  
 print(**"============ Best Spectral[Spectral] ============"**)  
 print(bestk[kc])  
 file.writelines(**"============ Best Spectral[Spectral] ============"** + **"\n"**)  
 file.writelines(str(bestk[kc]) + **"\n"**)  
 kc+=1  
  
 *# to do here: given this array of E\_outs in CV, find the max, its  
 # corresponding index, and its corresponding value of clusters* print(**"============ Best Spectral Array ============"**)  
 file.writelines(**"============ Best Spectral Array ============"** + **"\n"**)  
 print(bestk)  
 file.writelines(str(bestk) + **"\n"**)  
  
 print(**"============ Best Spectral ============"**)  
 file.writelines(**"============ Best Spectral ============"** + **"\n"**)  
 print(max(bestk))  
 file.writelines(str(max(bestk)) + **"\n"**)  
 print(**"============ Best Spectral Index ============"**)  
 file.writelines(**"============ Best Spectral Index ============"** + **"\n"**)  
 print(bestk.index(max(bestk)))  
 file.writelines(str(bestk.index(max(bestk))) + **"\n"**)  
  
 *# Calculates Running Time* run\_time = time.time() - start\_time  
 minutes, seconds = divmod(run\_time, 60)  
 hours, minutes = divmod(minutes, 60)  
 print(**"============ Running Time ============"**)  
 file.writelines(**"============ Running Time ============"** + **"\n"**)  
 print(**"Seconds Time Format --- %s seconds ---"** % (time.time() - start\_time))  
 print(**"Normal Time Format --- %d:%02d:%02d ---"** % (hours, minutes, seconds))  
 file.writelines(str(**"Seconds Time Format --- %s seconds ---"** % (time.time() - start\_time) + **"\n"**))  
 file.writelines(str(**"Normal Time Format --- %d:%02d:%02d ---"** % (hours, minutes, seconds)))

**data\_parser.py**

**import** re  
**import** xlwt  
**from** tempfile **import** TemporaryFile  
  
*# Forwards Dataset*forward = **"Dataset/forward\_json.txt"***# Discard Dataset*discard = **"Dataset/discard\_json.txt"***# Test Dataset*test\_data = **"Dataset/test\_json.txt"***# Debug Variable*debug = **False***# Dataset Excel Name***'''data\_sheet = xlwt.Workbook()  
sheet1 = data\_sheet.add\_sheet("test")'''***# Print to Excel***'''def print\_to\_excel(info\_list, sheet\_name):  
 for i,e in enumerate(info\_list):  
 sheet1.write(i,0,e)  
  
 data\_sheet.save(sheet\_name)'''***# Function to grab data from json***def** get\_data(file\_name):  
  
 data = open(file\_name).read()  
  
 *# Souce IP Information* src\_ip = re.findall(**'"src":\s"(\d+.\d+.\d+.\d+)"'**, data)  
 src\_port = re.findall(**'"src\_port":\s"(\d+)"'**, data)  
  
 *# Destination IP Information* dest\_ip = re.findall(**'"dest":\s"(\d+.\d+.\d+.\d+)"'**, data)  
 dest\_port = re.findall(**'"dest\_port":\s"(\d+)"'**, data)  
  
 *# Geolocation Information* city = re.findall(**'"city":\s"(.\*)",\s"host"'**, data)  
 subdivision = re.findall(**'"subdivision":\s"(.\*)",\s"name"'**, data)  
 lat = re.findall(**'"lat":\s"(-?\d+.\d+)",\s"country"'**, data)  
 country = re.findall(**'"country":\s"(.\*)",\s"postal"'**, data)  
 postal = re.findall(**'"postal":\s"(.\*)",\s"ASN"'**, data)  
 long = re.findall(**'"long":\s"(-?\d+.\d+)"}'**, data)  
  
 *# ISP Information* host = re.findall(**'"host":\s"(.\*)",\s"subdivision"'**, data)  
 host\_name = re.findall(**'"name":\s"(.\*)",\s"ip"'**, data)  
 isp\_ip = re.findall(**'"ip":\s"(\d+.\d+.\d+.\d+)",\s"lat"'**, data)  
 asn = re.findall(**'"ASN":\s"(\d+)",\s"long"'**, data)  
  
 *# Debug Messages* **if** debug:  
 print(**"src\_ip: "** + str(len(src\_ip)))  
 print(**"src\_port: "** + str(len(src\_port)))  
 print(**"dest: "** + str(len(dest\_ip)))  
 print(**"dest\_port: "** + str(len(dest\_port)))  
 print(**"city: "** + str(len(city)))  
 print(**"subdivision: "** + str(len(subdivision)))  
 print(**"lat: "** + str(len(lat)))  
 print(**"long: "** + str(len(long)))  
 print(**"country: "** + str(len(country)))  
 print(**"postal: "** + str(len(postal)))  
 print(**"host: "** + str(len(host)))  
 print(**"host\_name: "** + str(len(host\_name)))  
 print(**"isp\_ip: "** + str(len(isp\_ip)))  
 print(**"asn: "** + str(len(asn)))  
  
 **return** src\_ip, src\_port, dest\_ip, dest\_port, city, subdivision, lat, country, postal, long, host, host\_name, isp\_ip, asn  
  
*# Gets all the data and assigns it to the appropriate variable for printing later on*src\_ip, src\_port, dest\_ip, dest\_port, city, subdivision, lat, country, postal, long, host, host\_name, isp\_ip, asn = get\_data(discard)  
  
*#data\_types = [src\_ip, src\_port, dest\_ip, dest\_port, city, subdivision, lat, country, postal, long, host, host\_name, isp\_ip, asn]***def** get\_unique(list):  
 ulist = set(list)  
  
 **'''for value in ulist:  
 print(str(value))'''** new\_a = []  
  
 **for** ip **in** ulist:  
 new\_a.append(ip)  
  
  
 **return** new\_a  
  
*#get\_unique(src\_port)*unique\_list = get\_unique(src\_port)  
  
**for** i **in** range(len(src\_port)):  
 **for** j **in** range(len(unique\_list)):  
 **if** src\_port[i] == unique\_list[j]:  
 src\_port[i] = j\*10  
  
**for** stuff **in** src\_port:  
 print(stuff)  
*#--------------------------------#  
# Print Variables*src\_ip\_print = **False**src\_port\_print = **False**dest\_ip\_print = **False**dest\_port\_print = **False**city\_print = **False**subdivision\_print = **False**lat\_print = **False**country\_print = **False**postal\_print = **False**long\_print = **False**host\_print = **False**host\_name\_print = **False**isp\_ip\_print = **False**asn\_print = **False  
  
if** src\_ip\_print:  
 **for** ip **in** src\_ip:  
 print(ip)  
**if** src\_port\_print:  
 **for** port **in** src\_port:  
 print(port)  
**if** dest\_ip\_print:  
 **for** ip **in** dest\_ip:  
 print(ip)  
**if** dest\_port\_print:  
 **for** port **in** dest\_port:  
 print(port)  
**if** city\_print:  
 **for** city\_name **in** city:  
 print(city\_name)  
**if** subdivision\_print:  
 **for** subdivision\_name **in** subdivision:  
 print(subdivision\_name)  
**if** lat\_print:  
 **for** lat\_num **in** lat:  
 print(lat\_num)  
**if** country\_print:  
 **for** country\_name **in** country:  
 print(country\_name)  
**if** postal\_print:  
 **for** postal\_num **in** postal:  
 print(postal\_num)  
**if** long\_print:  
 **for** long\_num **in** long:  
 print(long\_num)  
**if** host\_print:  
 **with** open(**"temp.txt"**, **"w"**) **as** file:  
 **for** host\_url **in** host:  
 file.writelines(host\_url+**"\n"**)  
**if** host\_name\_print:  
 **with** open(**"temp.txt"**, **"w"**) **as** file:  
 **for** name **in** host\_name:  
 file.writelines(name + **"\n"**)  
**if** isp\_ip\_print:  
 **for** ip **in** isp\_ip:  
 print(ip)  
**if** asn\_print:  
 **for** num **in** asn:  
 print(num)  
*#--------------------------------#  
# Print All Records***'''for i in range(len(data\_types)):  
 for records in data\_types[i]:  
 print(records)'''**

***Bibliography***

[1] Hartigan, J. A., and M. A. Wong. “Algorithm AS 136: A K-Means Clustering Algorithm.” *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, 1979, pp. 100–108. [www.jstor.org/stable/2346830](http://www.jstor.org/stable/2346830).

[2] Hartigan, John A., and Manchek A. Wong. "Algorithm AS 136: A k-means clustering algorithm." *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28.1 (1979): 100-108.

[3] Ng, Andrew Y., Michael I. Jordan, and Yair Weiss. "On spectral clustering: Analysis and an algorithm." *Advances in neural information processing systems* 2 (2002): 849-856.

[4] Von Luxburg, Ulrike. "A tutorial on spectral clustering." *Statistics and computing* 17.4 (2007): 395-416.