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]

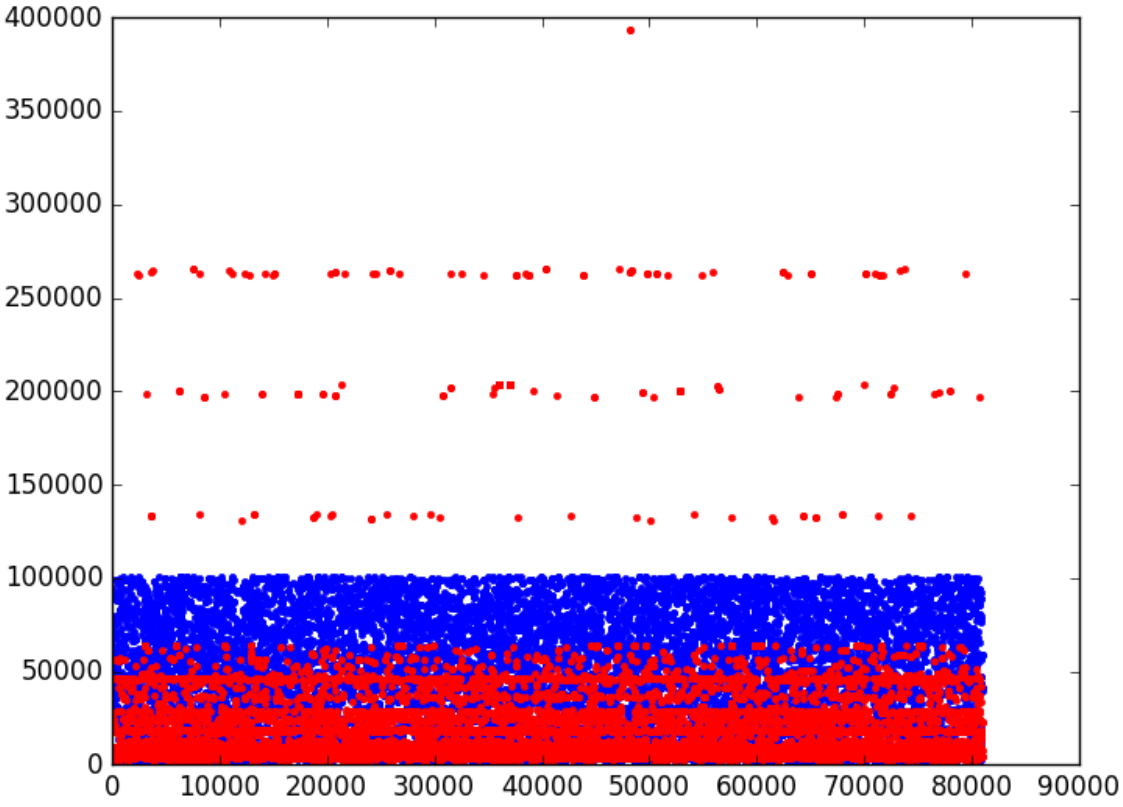
========== Best Spectral Score ==========

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.

[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.