Mode=Single

March 29, 2022

[1]:
```python
# MLFA Assignment 6
# Rohit Ranjan
# 20CS30066
```

[2]:
```python
#  import commands

import scipy.spatial
from scipy import stats
import numpy as np
import statistics
import pandas as pd
from tqdm import tqdm
import matplotlib.pyplot as plt
from collections import Iterable
from scipy.cluster.hierarchy import dendrogram
from sklearn.cluster import AgglomerativeClustering
```

```
/home/tfjuror/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:10:
DeprecationWarning: Using or importing the ABCs from 'collections' instead of
from 'collections.abc' is deprecated since Python 3.3,and in 3.9 it will stop
working
  # Remove the CWD from sys.path while we load stuff.
```

[3]:
```python
# loading csv
df = pd.read_csv('Twitter_data.csv')

# dropping rows with NaN
df = df.dropna(axis=0)

# shuffling dataframe
df = df.sample(frac=1)
```

```
/home/tfjuror/anaconda3/lib/python3.7/site-
packages/IPython/core/interactiveshell.py:3063: DtypeWarning: Columns
(0,1,3,4,9,11) have mixed types.Specify dtype option on import or set
low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

[4]:
```
# converting categorical to numerical
col_4 = {False: 0, True: 1}
col_3 = {'en': 0, '_u': 1}
days ={'Friday':1, 'Saturday':2, 'Sunday':3, 'Thursday':0}

df.iloc[:,4] = df.iloc[:,4].map(col_4)
df.iloc[:,3] = df.iloc[:,3].map(col_3)
df.iloc[:,1] = df.iloc[:,1].map(days)
```

[5]:
```
# train test split 90%-10%
df_test = df[-int(0.1*len(df)):]
df = df[:-int(0.1*len(df))]
print("Test set :",len(df))
print("Train set :",len(df_test))
```

```
Test set : 898
Train set : 99
```

[6]:
```
# droppping non-required columns
X_columns = [ ' Hour', ' Lang', ' IsReshare', ' Reach',
       ' RetweetCount', ' Likes', ' Sentiment', ' LocationID']
Y_columns = ['Day']

X = df[X_columns]
X.columns = X_columns
Y = df[Y_columns]

X_test = df_test[X_columns]
X_test.columns = X_columns
Y_test = df_test[Y_columns]
```

[7]:
```
print(X.shape)
print(Y.shape)
```

```
(898, 8)
(898, 1)
```

[8]:
```
# utility function to flatten nested lists
def flatten(lis):
     for item in lis:
         if isinstance(item, Iterable) and not isinstance(item, str):
             for x in flatten(item):
                 yield x
         else:
             yield item
```

[9]:
```
# utility function to find euclidean distance
def euclidean_dist(x,y):
    return np.sqrt(np.sum(np.square(x-y)))
```

[10]:
```
# utility function for train accuracy
def train_accuracy(X,Y,clusters,labels_list):
    correct=0
    for k,cluster in enumerate(clusters):
        for idx in cluster:
            if(Y.iloc[idx,0]==labels_list[k]):
                correct+=1
    return correct/len(X)
```

[11]:
```
# utility function for test accuracy
def test_accuracy(X_test,Y_test,X,Y,clusters,labels_list):
    correct=0
    cluster_centres=[]
    label_map = np.zeros((len(X),))
    pred_labels=[]
    for k,cluster in enumerate(clusters):
        arr = np.array([X.iloc[idx,:] for idx in cluster])
        mean = np.mean(arr,axis=0)
        cluster_centres.append(mean)
        for idx in cluster:
            label_map[idx]=labels_list[k]
    test_dist = scipy.spatial.distance.cdist(X_test,X)
    for i in range(len(X_test)):
        closest = np.argmin(np.array(test_dist[i]))
        pred_labels.append(label_map[closest])
        if(label_map[closest]==Y_test.iloc[i,0]):
            correct+=1
    return correct/len(X_test),pred_labels,label_map,cluster_centres
```

[12]:
```
# mutual distances array calculated only once very efficiently
dist_arr = scipy.spatial.distance.cdist(X,X)
for i in range(len(X)):
    dist_arr[i,i]=1e9
```

[13]:
```
# dictionary to store our dendogram
levels = {}
```

[14]:
```
# initialising all data points as separate clusters
clusters =[]
for i in range(len(X)):
    clusters.append([i])
levels[str(len(clusters))] = clusters.copy()
```

[15]:
```python
# calculating the complete tree with single linkage
def generator():
    while len(clusters)>1:
        yield

for _ in tqdm(generator()):
    curr_arr = np.zeros((len(clusters),len(clusters)))
    for i in range(len(clusters)):
        for j in range(len(clusters)):
            if(i==j):
                curr_arr[i,j]=1e9
                continue
            min=1e9
            for idx_i in clusters[i]:
                for idx_j in clusters[j]:
                    if(min>dist_arr[idx_i,idx_j]):
                        min=dist_arr[idx_i,idx_j]
            curr_arr[i,j]=min

    prev_clusters = clusters.copy()
    i,j = np.argwhere(curr_arr == np.min(curr_arr))[0]
    clusters.pop(i)
    if(i!=j):
        if(i>j):
            clusters.pop(j)
        else:
            clusters.pop(j-1)
    clusters.append(list(flatten([prev_clusters[i],prev_clusters[j]])))

    levels[str(len(clusters))] = clusters.copy()
```

```
897it [07:49,  1.91it/s]
```

[16]:
```python
# choose a level to cut
level_cut = 100
```

[17]:
```python
clusters = levels[str(level_cut)]
labels_list=[]
for cluster in clusters:
    labels = np.array([Y.iloc[i,0] for i in cluster])
    label = stats.mode(labels)[0]
    labels_list.append(label)
```

[18]:
```python
print('Train Accuracy: ',train_accuracy(X,Y,clusters,labels_list))
```

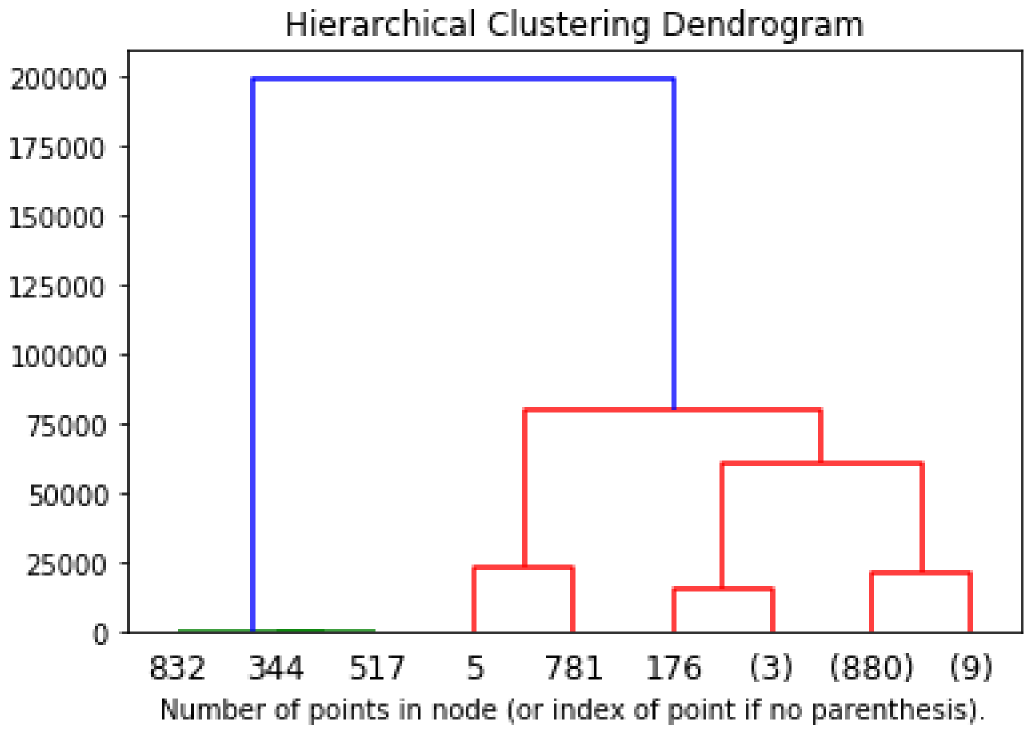
```
Train Accuracy:  0.48552338530066813
```

[19]:
```
test_acc,pred_labels,label_map,cluster_centres =␣
↪test_accuracy(X_test,Y_test,X,Y,clusters,labels_list)
print('Test Accuracy: ',test_acc)
```

```
Test Accuracy:  0.36363636363636365
```

[20]:
```
def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, counts]
    ).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)


model = AgglomerativeClustering(distance_threshold=0, n_clusters=None,␣
↪linkage='single')
model = model.fit(X,Y)
```

[21]:
```
# Visualising the Dendogram
plt.title("Hierarchical Clustering Dendrogram")
# plot the top three levels of the dendrogram
plot_dendrogram(model, truncate_mode="level", p=3)
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.show()
```



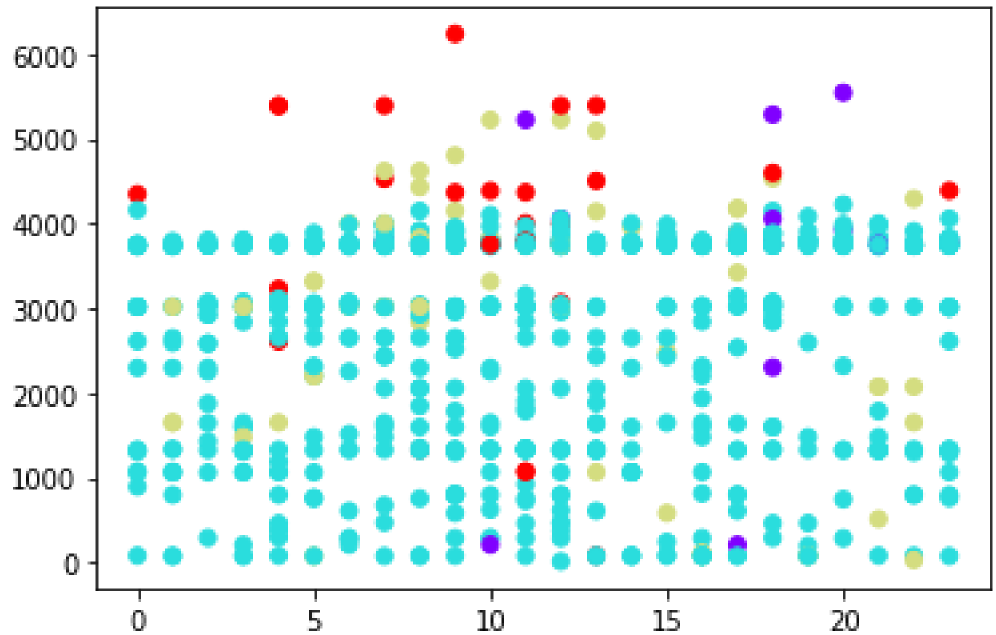
[22]:
```
# We can see that our clusters are the exact same as the scipy package
print(levels['3'])
```

```
[[832, 344, 517], [5, 781], [176, 108, 428, 557, 405, 236, 707, 19, 313, 476,
750, 802, 49, 390, 489, 612, 63, 649, 808, 549, 746, 370, 509, 6, 690, 221, 582,
548, 560, 777, 439, 493, 561, 632, 62, 703, 378, 494, 172, 254, 820, 878, 368,
584, 855, 248, 739, 310, 845, 275, 693, 46, 819, 31, 841, 34, 68, 527, 552, 511,
535, 113, 181, 426, 280, 524, 718, 269, 228, 291, 478, 760, 346, 461, 84, 287,
721, 270, 773, 546, 610, 667, 646, 7, 182, 882, 887, 785, 866, 265, 447, 518,
309, 695, 162, 268, 653, 354, 479, 483, 709, 9, 104, 158, 340, 792, 242, 825,
277, 814, 336, 544, 726, 192, 793, 205, 339, 597, 435, 818, 231, 387, 758, 20,
608, 374, 661, 373, 114, 685, 239, 588, 717, 132, 302, 890, 743, 806, 210, 106,
446, 453, 57, 590, 204, 396, 98, 53, 568, 177, 450, 276, 771, 485, 66, 316, 381,
140, 441, 681, 538, 256, 579, 41, 827, 216, 798, 250, 879, 452, 477, 706, 161,
260, 244, 432, 515, 139, 616, 829, 107, 409, 633, 671, 565, 660, 501, 688, 13,
628, 266, 784, 589, 348, 135, 539, 751, 39, 480, 573, 208, 861, 81, 800, 211,
44, 209, 45, 735, 215, 554, 226, 704, 731, 297, 24, 294, 327, 225, 596, 722,
618, 101, 816, 630, 787, 74, 186, 413, 471, 567, 627, 47, 163, 389, 83, 725,
520, 824, 398, 644, 142, 765, 60, 472, 168, 227, 122, 238, 422, 607, 150, 663,
715, 331, 523, 23, 840, 431, 375, 148, 883, 834, 18, 93, 332, 380, 643, 311,
127, 631, 757, 173, 724, 496, 763, 780, 449, 320, 563, 230, 886, 666, 775, 371,
307, 668, 499, 255, 697, 87, 611, 15, 850, 762, 502, 408, 865, 138, 796, 891,
382, 894, 786, 247, 54, 315, 602, 815, 640, 220, 732, 212, 338, 531, 436, 712,
571, 683, 326, 30, 648, 404, 25, 59, 529, 895, 67, 791, 343, 679, 486, 839, 553,
```

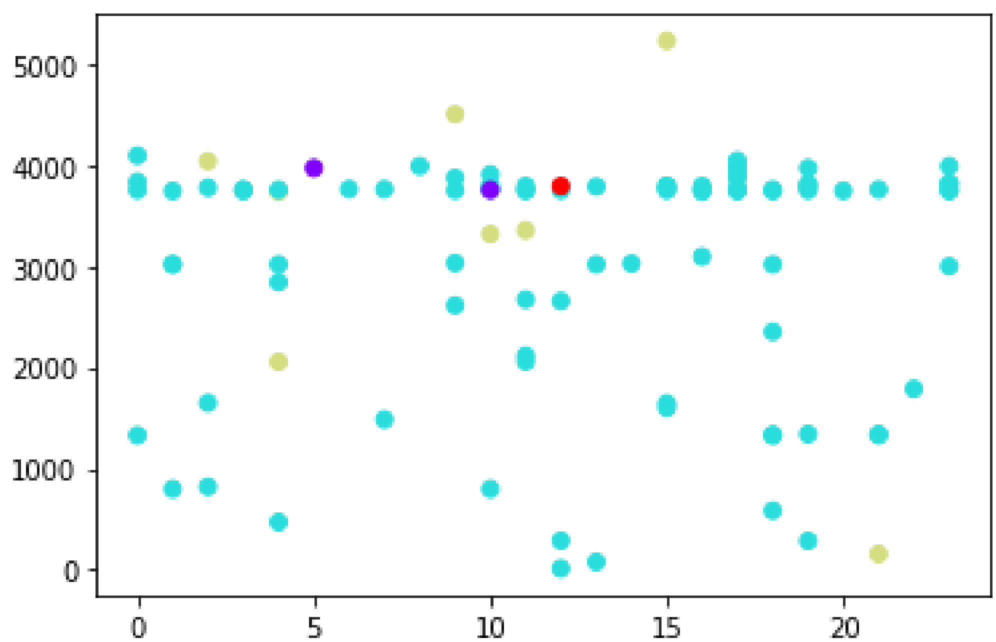
```
82, 262, 738, 259, 730, 165, 55, 58, 684, 748, 123, 675, 637, 481, 271, 323,
581, 512, 776, 445, 500, 533, 451, 747, 393, 3, 801, 65, 94, 884, 888, 218, 399,
889, 629, 728, 0, 716, 290, 42, 99, 532, 872, 733, 585, 304, 164, 369, 687, 240,
789, 848, 525, 88, 664, 214, 795, 759, 306, 357, 719, 506, 860, 621, 487, 599,
257, 713, 305, 665, 434, 324, 843, 170, 455, 96, 587, 766, 545, 740, 755, 91,
137, 415, 770, 547, 893, 29, 52, 178, 772, 711, 200, 559, 856, 85, 152, 555,
600, 838, 652, 638, 10, 388, 720, 207, 272, 126, 598, 708, 862, 442, 805, 333,
670, 246, 337, 846, 874, 206, 243, 70, 615, 877, 75, 274, 153, 288, 406, 28,
159, 700, 692, 125, 764, 456, 454, 778, 366, 752, 826, 103, 188, 347, 328, 803,
407, 783, 619, 622, 634, 425, 503, 32, 448, 578, 691, 562, 319, 424, 753, 680,
830, 864, 411, 857, 482, 100, 167, 284, 258, 880, 614, 76, 189, 358, 574, 143,
443, 698, 821, 844, 97, 444, 577, 124, 264, 419, 761, 261, 300, 617, 26, 586,
809, 551, 156, 400, 360, 465, 491, 870, 229, 831, 71, 166, 835, 196, 298, 421,
73, 460, 723, 414, 403, 516, 522, 714, 836, 80, 702, 128, 645, 36, 50, 430, 349,
858, 744, 86, 756, 847, 492, 379, 179, 566, 171, 237, 151, 263, 252, 677, 21,
273, 112, 183, 657, 641, 701, 505, 279, 892, 689, 729, 222, 423, 470, 4, 56,
121, 174, 412, 191, 682, 469, 145, 440, 462, 672, 656, 397, 376, 350, 293, 592,
356, 513, 51, 253, 416, 508, 219, 504, 296, 281, 605, 14, 417, 314, 361, 674,
420, 804, 241, 741, 854, 595, 105, 33, 863, 352, 286, 530, 655, 651, 335, 427,
154, 217, 669, 550, 737, 678, 110, 35, 896, 438, 580, 556, 868, 198, 521, 193,
794, 1, 383, 233, 299, 11, 129, 570, 594, 686, 89, 119, 514, 72, 782, 736, 642,
572, 822, 650, 849, 199, 569, 823, 623, 283, 365, 79, 322, 367, 131, 593, 391,
321, 859, 851, 111, 474, 130, 852, 303, 811, 292, 289, 788, 203, 362, 120, 528,
40, 27, 745, 459, 540, 604, 575, 22, 317, 146, 312, 351, 519, 543, 875, 144,
749, 705, 658, 769, 458, 133, 727, 639, 61, 673, 201, 591, 564, 624, 202, 807,
285, 213, 676, 696, 102, 136, 2, 345, 224, 467, 620, 12, 402, 833, 395, 457,
842, 871, 537, 195, 625, 175, 392, 353, 118, 92, 377, 301, 799, 810, 341, 534,
234, 318, 249, 613, 295, 235, 77, 197, 185, 754, 48, 558, 17, 662, 475, 742,
497, 779, 180, 881, 401, 359, 464, 654, 797, 355, 774, 386, 251, 330, 463, 484,
813, 364, 576, 69, 626, 869, 78, 223, 606, 873, 897, 384, 334, 433, 541, 394,
635, 828, 329, 768, 429, 468, 418, 837, 116, 149, 603, 410, 495, 817, 16, 812,
694, 699, 601, 134, 190, 790, 473, 385, 710, 526, 437, 194, 147, 867, 157, 169,
187, 282, 363, 278, 64, 609, 117, 155, 141, 325, 498, 647, 885, 342, 466, 90,
160, 542, 8, 734, 37, 38, 43, 853, 767, 659, 876, 184, 267, 372, 507, 95, 245,
536, 115, 308, 109, 488, 490, 583, 510, 232, 636]]
```

[23]:

```
# Scatter plot of 0th and 7th attribute along with predicted labels for train␣
↪set
plt.scatter(X.iloc[:,0],X.iloc[:,7],c=label_map,cmap='rainbow')
plt.show()
```

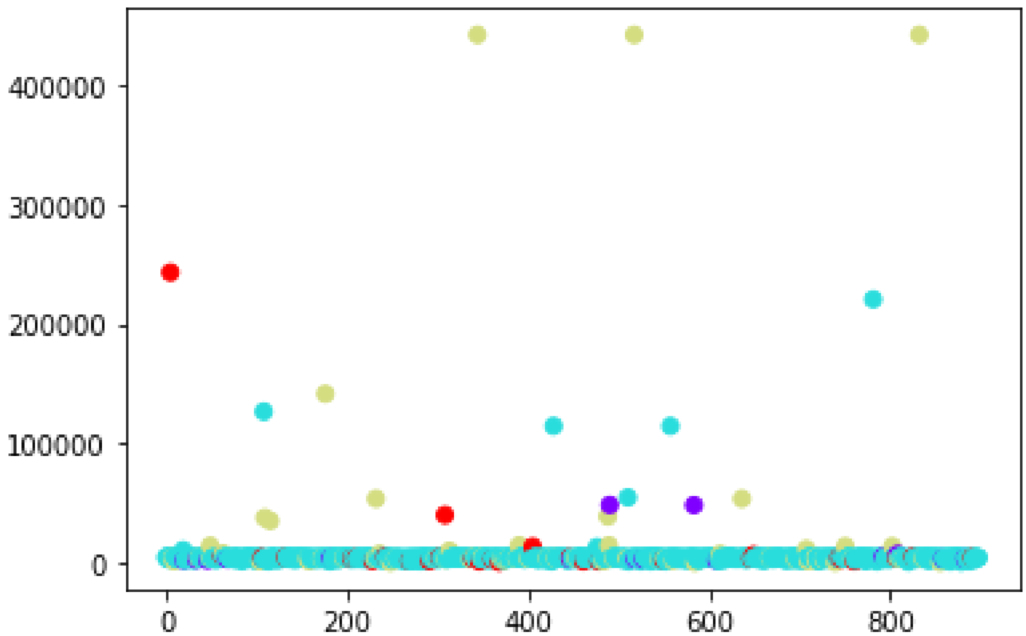


[24]:
```
# Scatter plot of 0th and 7th attribute along with predicted labels for test set
plt.scatter(X_test.iloc[:,0],X_test.iloc[:,7],c=pred_labels,cmap='rainbow')
plt.show()
```

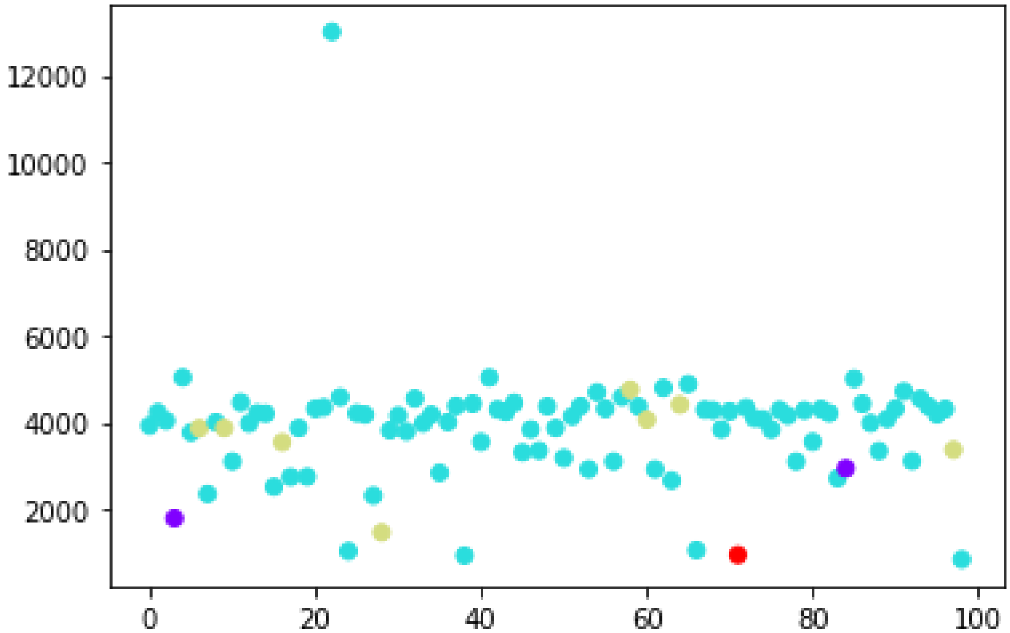


[25]:
```
X_test_dist = [euclidean_dist(X_test.iloc[i,:],np.mean(X,axis=0)) for i in␣
↪range(len(X_test))]
X_dist = [euclidean_dist(X.iloc[i,:],np.mean(X,axis=0)) for i in range(len(X))]
```

[26]:
```
# Scatter plot of distance of datapoint from mean along with predicted labels␣
↪for train set
plt.scatter([i for i in range(len(X))],X_dist,c=label_map,cmap='rainbow')
plt.show()
```



[27]:
```
# Scatter plot of distance of datapoint from mean along with predicted labels␣
↪for test set
plt.scatter([i for i in␣
↪range(len(X_test))],X_test_dist,c=pred_labels,cmap='rainbow')
plt.show()
```


12000
10000
8000
6000
4000
2000
0
20
40
60
80
100

```
[ ]:
```