Assignment 3: Clustering

Elmar Hajizada

1 Task 1

As a first task, we will show the distribution of ϕ and ψ-angles observed in proteins as a scatter plot and as a heat map. To get the angles, we first store all the protein data given as part of the task and pick the angles out from it.

```
full=pd.read_csv('data_all.csv')
plotValues(full)
def plotValues(df):
    #Read in the data
    phi = df['phi']; psi = df['psi']
```

The angles are then plotted in two subplots. The first uses scatter to show the angles as singular points, The other uses hist2d to display the density of points within each pair of integer degrees.

```
    #Create a scatter plot
    plt.scatter(df['phi'],df['psi'],s=0.1)
    plt.axis([-180, 180, -180, 180])
    plt.xticks(np.linspace(-180,180,13),('-180','-150','-120','-90','-60','-30','0','30','60','90','120↩
        ','150','180'))
    plt.yticks(np.linspace(-180,180,13),('-180','-150','-120','-90','-60','-30','0','30','60','90','120↩
        ','150','180'))
    plt.xlabel('phi (degrees)')
    plt.ylabel('psi (degrees)')
    plt.show()
    #Plots the heat map when the window is closed
    plt.clf()
    #Calculate the density
    plt.hist2d(phi,psi,bins=(360,360),cmap='rainbow')
    plt.colorbar()
    plt.xticks(np.linspace(-180,180,13),('-180','-150','-120','-90','-60','-30','0','30','60','90','120↩
        ','150','180'))
    plt.yticks(np.linspace(-180,180,13),('-180','-150','-120','-90','-60','-30','0','30','60','90','120↩
        ','150','180'))
    plt.xlabel('phi (degrees)')
    plt.ylabel('psi (degrees)')
    plt.show()
```

The scatter-plot and heat map can be seen in the figures 1 and 2, respectively.
In figure 1, we can immediately see three or four larger clusters. Two large ones in the top left and middle left, a smaller one in the middle right, and an even smaller one in the bottom left. It should be noted that since phi and psi are both periodic by 360, the cluster in the bottom left should in theory belong to the one in the top left. From figure 2, we can see that the points in the two larger clusters are also very dense, especially the ones in the middle left one. We also see some hints of the cluster on the right, but compared to the other two, it seems very faint.

2 Task 2

In this task we tried to use K-Means clustering to cluster our data. So in order to cluster the data we need to find optimal K value for the data. So in order to find optimal K value we have used "Elbow" method

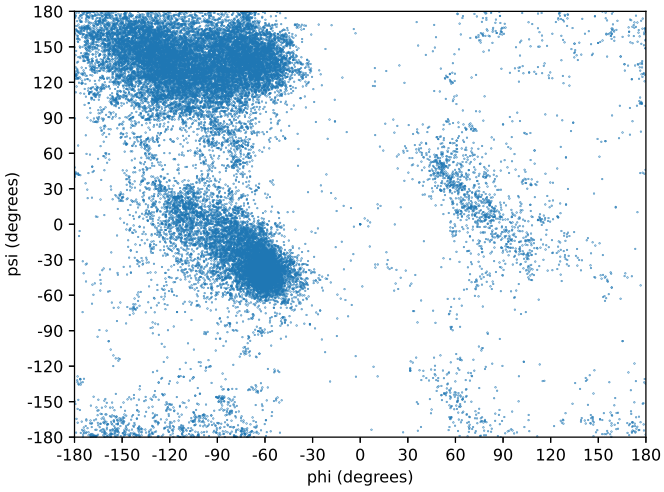

Figure 1: A scatter-plot of the angles observed in proteins

where we took k clusters between 1 and 10. The idea is to experiment with various K clusters and find optimal one. We want to have clusters as few as possible also that each cluster is as tight as possible. So graphically finding an elbow point is obvious sometimes, however depending on data this can be changed. Sometimes it is not obvious graphically to distinguish elbow point. Therefore, we need to find elbow point mathematically. In order to do this, first we found wss (sum of all squared errors within cluster) scores for K clusters in range 1 - 10.

```
def calculate_WSS(points, kmax):
  sse = []
  for k in range(1, kmax+1):
    kmeans = KMeans(n_clusters = k).fit(points)
    centroids = kmeans.cluster_centers_
    pred_clusters = kmeans.predict(points)
    curr_sse = 0

    # calculate square of Euclidean distance of each point from its cluster center and add to current ↩
        WSS
    for i in range(len(points)):
      curr_center = centroids[pred_clusters[i]]
      curr_sse += (points[i, 0] - curr_center[0]) ** 2 + (points[i, 1] - curr_center[1]) ** 2
    sse.append(curr_sse)
  return sse
```

Now we have all WSS scores for each clusters. Then, we need to find which of these clusters is the optimal one. We are looking for the optimal point that indicates balance between greater homogeneity within the cluster and greater difference between the clusters. Thanks to the math we can find this point. WSS scores for each clusters has been shown on Figure 3 and we have to calculate distance between a point which is k cluster points and line which connects first and last clusters. The point that has highest distance between the line is the optimal K value.

We have used following math formula to find out this distance:

$$distance(P_0, P_1, (x, y)) = \frac{|(y_1 - y_0)x - (x_1 - x_0)y + x_1y_0 + x_0y_1|}{\sqrt{(y_1 - y_0)^2 + (x_1 - x_0)^2}}$$

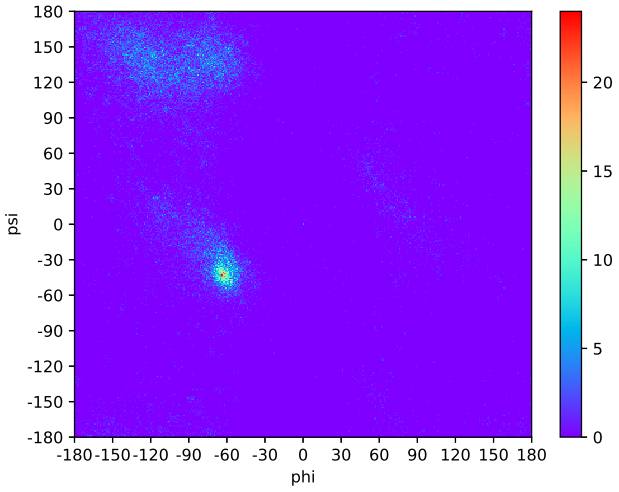

Figure 2: A heat map of the angles observed in proteins

So in our case P_0 and P_1 are points whose cluster value k is 1 and 10 respectively. Point (x, y) can be coordinates of any point that we want to calculate distance to the line.

```
def optimum_k(k_clusters, wcss):
    x1, y1 = 2, wcss[0] # first point
    x2, y2 = 20, wcss[-1] # last point

    distances = []
    for i in range(len(wcss)):
        x0 = k_clusters[i]
        y0 = wcss[i]
        numerator = abs((y2-y1)*x0 - (x2-x1)*y0 + x2*y1 - y2*x1)
        denominator = np.sqrt((y2 - y1)**2 + (x2 - x1)**2)
        distances.append(numerator/denominator)

    return distances.index(max(distances)) + 1
```

By using this function we can get distances of any cluster points to the line. The cluster point that has highest distance will be our optimal K cluster.

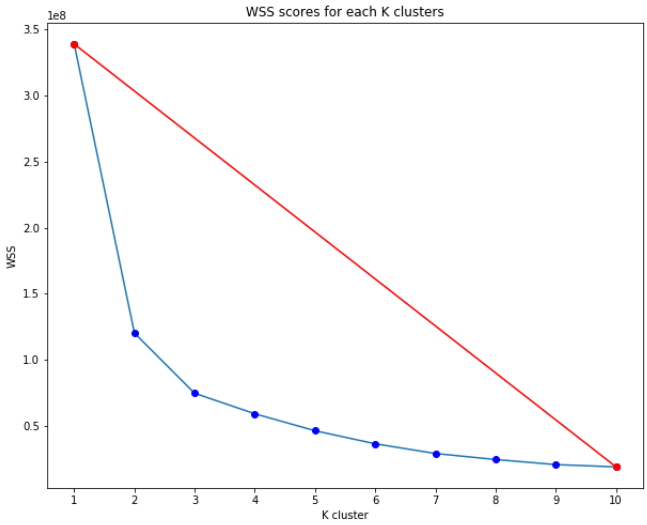

Figure 3: WSS scores for each K clusters

K clusters	1	2	3	4	5	6	7	8	9	10
distances	1.0	12.3	13.86	13.74	13.46	12.87	12.43	11.68	10.9	10.0

Figure 4: Distances between clusters points and a line

After applying this function, we got optimal K value as 3. So by using elbow method we concluded that there optimally should be 3 clusters for K means algorithm. As we can see from the Figure 4, we can say that the highest distance that cluster has between its point and the line, is 3. So therefore, we can consider optimal k value for k means as 3.

```
def kmeans_clusterplot(X, k_clusters):
    kmeans = KMeans(n_clusters=k_clusters, random_state=0).fit(X)
    y_kmeans = kmeans.predict(X)
    plt.figure(figsize=(8, 6))
    plt.scatter(X[:, 1], X[:, 0], c=y_kmeans, s=5, cmap='viridis')
    centers = kmeans.cluster_centers_
    plt.scatter(centers[:, 1], centers[:, 0], c='red', s=30, alpha=0.5)
    plt.xlabel('phi')
    plt.ylabel('psi')
```

As we see in following figure, there are 3 clusters whose main cluster points were shown by red color. These points are reasonable since whole data clustered into 3 clusters and K in our case is an optimal cluster value for this data. However, as can be seen from Figure 5 there also noises that we can call outliers in this data which is not taken into account by K-Means algorithm. One of the drawbacks of this algorithm is that, it only requires to specify number of K clusters and it will assign each instance to its nearest cluster. So in this case, outliers will be also assigned to its corresponding cluster. In the task 3 we will discuss how DBSCAN can work with this issue.

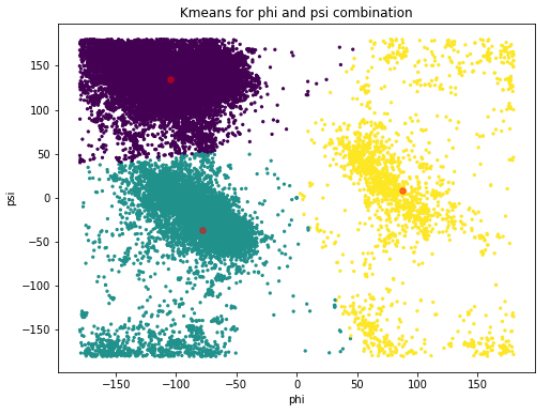

Figure 5: Kmeans for psi and phi combination

Furthermore in order to have much nicer visualizations of the data using K means we tried to shift some of our data points. As we can see from Figure 5 some of the data points gathered at the border parts of the scatter plot where phi values is less than 0 and or psi values is less -150, on the other hand if we try to change values of these numbers, we can get better visualization. Since the phi and psi values are peroidic angles, we can define these angles in another form. For instance, if phi value for the instance is -170, it is the same thing with $360 + (-170) = 190$, there is only difference in direction of the angles. But in both of the examples, element will end up in the same place. So we shifted phi values which are below 0 and also psi values which are less than -150 in order to have better visualization. As we can see from Figure 7, we ended

up by 3 clusters which was optimal K value obtained through elbow method after shifting the elements.

k clusters	1	2	3	4	5	6	7	8	9	10
sum of squared errors for original data	338745899.88	120313517.05	74759869.22	59268161.72	46432195.42	36577586.9	29215136.98	24673341.58	20833124.36	19046470.41
sum of squared errors for shifted data	302788159.82	117153759.0	58879307.86	43321444.09	35492329.8	30145934.42	25734139.59	22634829.96	20581690.44	18611807.08

Figure 6: Sum of squared errors for original and shifted data corresponding to each K clusters

As you can see from Figure 6, we can easily say that after shifting the data, wss scores which is sum of all squared errors have been decreased for each K clusters. Moreover, we can observe a dramatic decrease in wss error for optimal K cluster which is 3 for both original and shifted data. From this table we can say that the shifting the psi values below 0 and phi values below -150 is a reasonable choice, since wss scores for each cluster decreased and we also get much better view visually as in Figure 6.

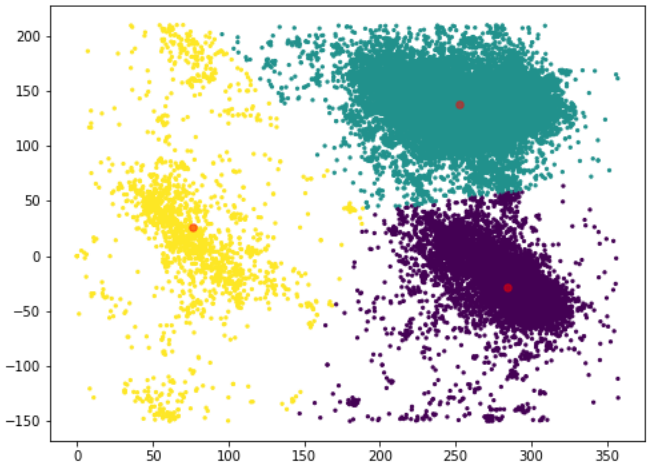

Figure 7: Kmeans with shifted data for psi and phi combination

3 Task 3

Next, we tried to cluster the data using the DBSCAN cluster model, this time starting with the shifted data. In order to find a good ϵ, the distance of samples belonging to the same neighborhood, we used something known as the euclidean distance method. To do this, we calculate the minimal distance between a point and its n nearest neighbors (Here we chose n to be 10). Then, we sort the distances in ascending order and plot them, giving us an elbow curve that lets us find the optimal ϵ. For this we created the following code,

```
def findOpteps(df):
    #Read the angles
    angles=df.filter(items = ['phi', 'psi'])
    data=angles.to_numpy()
    #Find the distance between the nearest neighbors
    neigh = nbors.NearestNeighbors(n_neighbors=10).fit(data)
    distances, indices = neigh.kneighbors(data)
    #Sort the distances
    distances = np.sort(distances, axis=0)
    #Plot them
    distances = distances[:,1]
    plt.plot(distances)
    plt.show()
```

Which gives us figure 8. By looking at the y-axis near the curviest part of the plot, we can see that the optimal ϵ value is somewhere around 1.9.

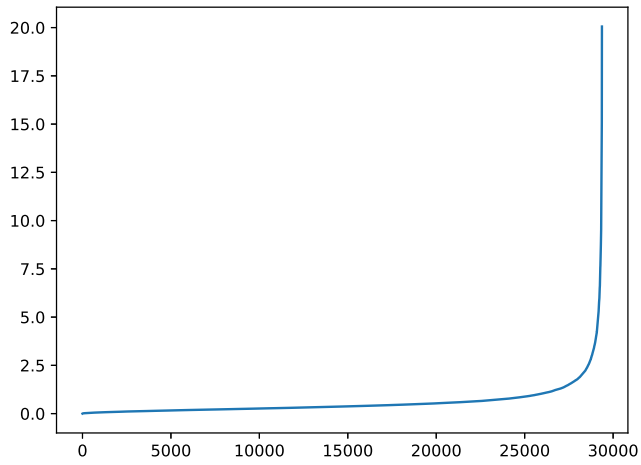

Figure 8: Plot of the minimal distance between points and their 10 nearest neighbors sorted. The optimal ϵ is found where the curvature is the largest

To choose the minimum number of neighbors for a point to be considered a core point, we simply performed DBSCAN with various number of neighbors and chose one that minimized the number of small clusters while letting the smaller noticeable cluster be considered a cluster.

```
def dbscan(df,eps,samples):
    #Read the angles
    angles=df.filter(items = ['phi', 'psi'])
    data=angles.to_numpy()
    #Cluster the data
    clustering = DBSCAN(eps=eps, min_samples=samples).fit(data)
    labels=clustering.labels_
    #Estimate the number of clusters and amount of noise
    n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
    n_noise_ = list(labels).count(-1)
    #Separate the noisy data from the unnoisy data.
    unnoisy_data = df[labels!=-1]
    noisy_data = df[labels==-1]
    print("Estimated number of clusters: %d" % n_clusters_)
    print("Estimated amount of noise: %d" % n_noise_)
    #Make a scatter plot of the clustered data.
    plt.scatter(unnoisy_data['phi'],unnoisy_data['psi'],c=labels[labels!=-1], cmap='Spectral',s=1)
    plt.scatter(noisy_data['phi'],noisy_data['psi'],c='black',s=1)
    plt.axis([-180, 180, -180, 180])
    plt.xticks(np.linspace(-180,180,13),('-180','-150','-120','-90','-60','-30','0','30','60','90','120↩
        ','150','180'))
    plt.yticks(np.linspace(-180,180,13),('-180','-150','-120','-90','-60','-30','0','30','60','90','120↩
        ','150','180'))
    plt.xlabel('phi')
    plt.ylabel('psi')
    plt.show()
```

In the end, we chose the minimal samples to be 6, which gave us the clusters in figure 9.

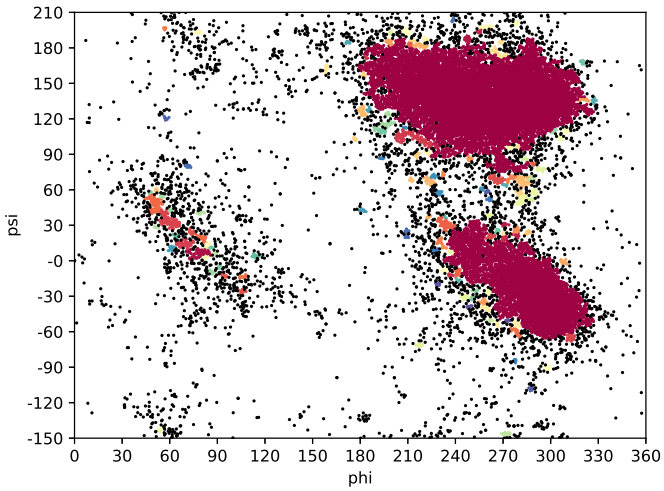

Figure 9: Scatter plot of the clusters found in the proteins with black noise using DBSCAN with $\epsilon = 1.9$ and min_samples= 6

We can see that while the two larger clusters are treated as separate clusters, the smaller noticeable cluster becomes a number of smaller clusters. The same thing happens to parts of the larger clusters as well, and some small clusters outside that is found leading to a total of 138 clusters formed.

As for the noise, there are a total of 3745 nodes that are classified as noise. By looking at the data that has been labelled as noise, we can see which types of protein are the most frequent outliers.

```
def plot_noise(df,eps,samples):
    #Read in the angles
    angles=df.filter(items = ['phi', 'psi'])
    data=angles.to_numpy()
    #Cluster the data
    clustering = DBSCAN(eps=eps, min_samples=samples).fit(data)
    labels=clustering.labels_
    noisy_data = df[labels==-1]
    #Get all proteins that are considered noise
    noisy_proteins = noisy_data['residue name']
    #Calculate the frequency of each label
    amount = noisy_proteins.value_counts(sort=True)
    #Plot the frequency of the labels
    plt.clf()
    plt.bar(amount.keys(),amount)
    plt.show()
```

The result of this can be found in figure 10, where we can see that the by far most frequent outlier is the protein GLY with over 1000 points being classified as noise. After this, the next most frequent outlier has around 300 points classified as noise, with most other proteins having a similar amount of noise. This means that we would likely get completely different clusters if we were to just look at GLY.

If we were to compare the results from clustering with K-means and DBSCAN, we can see that with K-means clustering, we get a small number of clusters for a large amount of data which also seems to mostly correspond to the clusters that we can see from just looking at the data. DBSCAN, in comparison has a larger number of clusters and does not fully encapsulate the smaller, but noticable clusters. On the other hand, DBSCAN knows when some of the data is just noise, unlike K-means which tries to cluster every point in the data, including points that are far from the rest of the data. This is likely because with K-means, we specify the number of clusters and let the algorithm figure out how they should be clustered, while for DBSCAN, we specify what a cluster should mean, and let the algorithm create the clusters.

While we have seen that we can create clusters using DBSCAN, the question is whether the clusters will remain using small shifts to either ϵ or the sample limit of the algorithm. To test this, we plotted two new DBSCANs, one where we changed ϵ to be 1.8 and one where the minimal number of samples were reduced to 5. The results of each of these scans are found in figure 11 and 12, respectively.
If we compare both of these figures with figure 9, there seems to be little difference at a glace. However, a bunch of smaller clusters disappear when ϵ decreases while small parts of the larger clusters split to become small clusters or just noise. This gives us about 300 more points classified as noise. When we decrease the minimal samples on the other hand, some new small clusters appear, giving us 181 total clusters compared to the 138 from before, while simultaneously making the smaller noticeable cluster consist of fewer clusters. The majority of the large clusters is still intact though, meaning that small clusters are not robust to changes, while large ones should be robust.

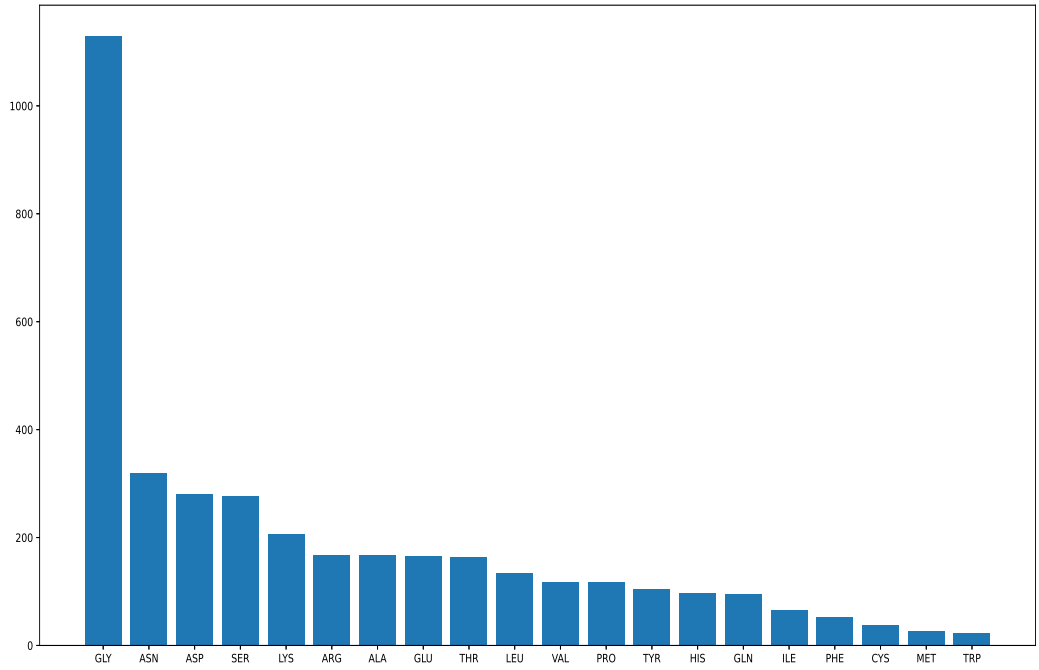

Figure 10: Bar graph of the frequency of a protein being labelled as noise

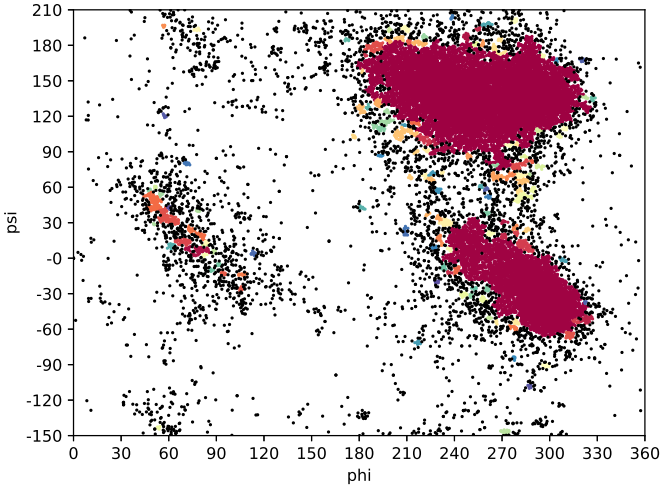

Figure 11: Scatter plot of the clusters found in proteins using DBSCAN with $\epsilon = 1.8$ and min_samples= 6

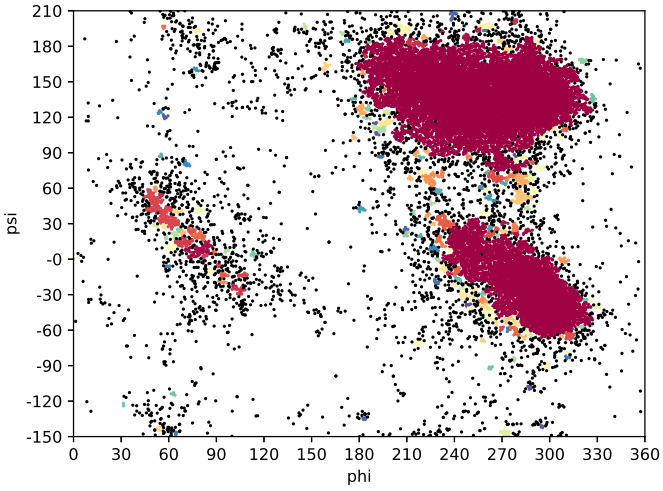

Figure 12: Scatter plot of the clusters found in proteins using DBSCAN with $\epsilon = 1.9$ and min_samples= 5

4 Task 4

Since the residue type of each protein is given, we can look at the clustering of a type of protein and see whether there are any differences between these and the clusters of all proteins. In this task, we are going to look at the proteins PRO and GLY, first using DBSCAN, and then by using K-means.

To plot the two proteins using DBSCAN, we must once again find a good ϵ to use. To do this, we use the exact same method as before, we sort the distance to a points 10 nearest neighbors, and pick the distance where the graph has the largest curve. These distances are seen in figure 13 for the protein PRO, and in figure 14 for the protein GLY.

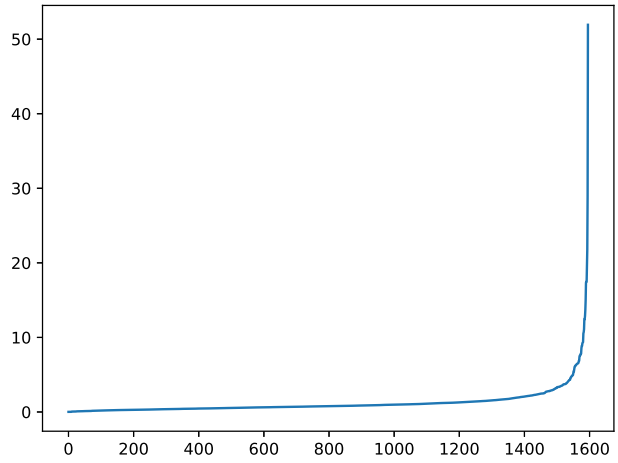

Figure 13: Plot of the minimal distance between points and their 10 nearest neighbors for the protein PRO

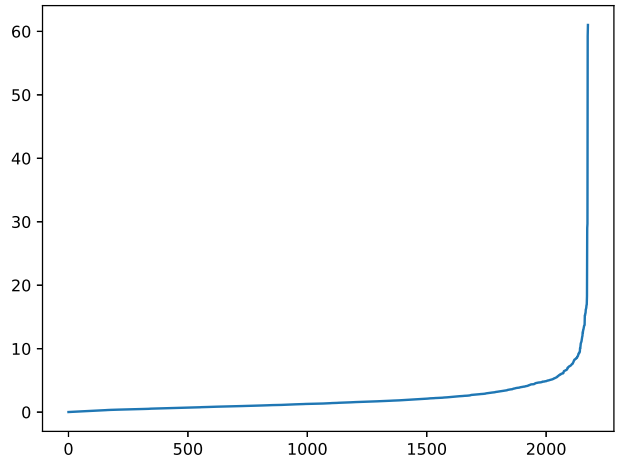

Figure 14: Plot of the minimal distance between points and their 10 nearest neighbors for the protein GLY

From these, we easily see that ϵ should be about 4.7 for PRO, and about 9 for GLY. Next, we use DBSCAN with the given ϵ and with min_samples being set to 8 to cluster both proteins. The clustering of the proteins PRO and GLY can be found in figures 15 and 16, respectively.

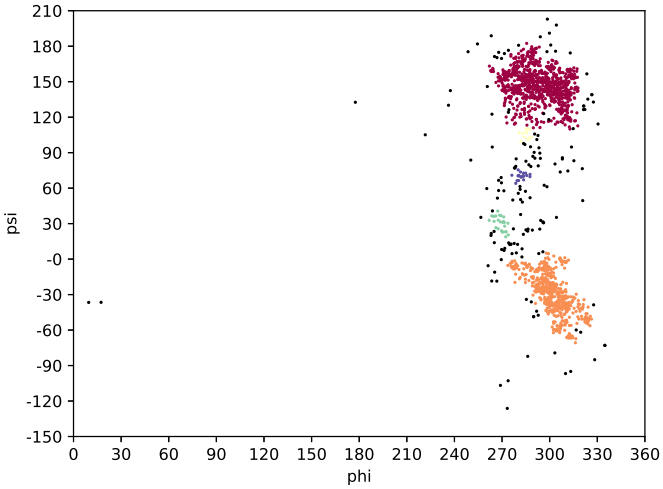

Figure 15: Scatter plot of the clusters found in PRO using DBSCAN with $\epsilon = 4.7$ and min_samples $= 8$

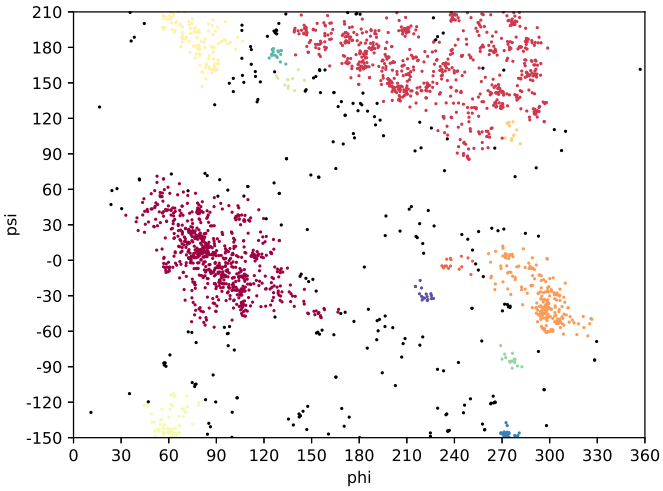

Figure 16: Scatter plot of the clusters found in GLY using DBSCAN with $\epsilon = 9$ and min_samples $= 8$

From the figures, we can see that the clusters found for PRO are mostly contained in the two larger clusters that we've seen from the other proteins. While GLY also contains these clusters, it also contains a lot of points from the smaller cluster, as well as a noticeable cluster when $\phi \approx 90$ and $\psi \approx 180$ that we've previously only considered to be noise, which we could observe before when we saw that a lot of the points of GLY were classified as noise before. A conclusion that can be drawn from this is that with stratification, we might find clusters of particular proteins that would not otherwise be found.

Now we are going to analyze PRO, GLY and general clusters using K means. In order to differentiate clusters of GLY, PRO and whole data, we need to find again optimal K value for GLY and PRO residue types.
Firstly, we started again find an optimal value for residue type PRO by using Elbow method. Graphically also algebraically we found out that optimal value for K is 2.

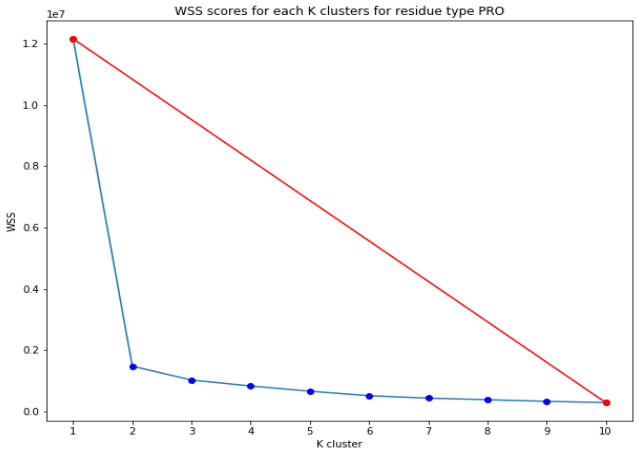

Figure 17: plot of elbow point method for PRO residue type

On the other hand, we also find an optimal K value for residue type GLY which was 4 by using again elbow method.

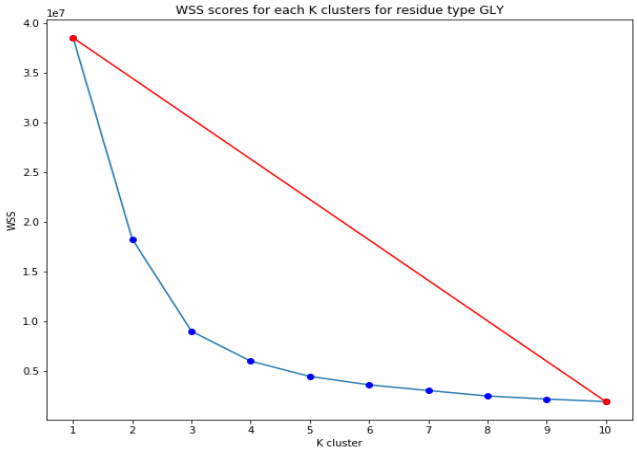

Figure 18: plot of elbow point method for GLY residue type

After obtaining optimal K values for both PRO and GLY residue types, we can plot our K-means clusters to see where PRO and GLY spreaded in our data and what differences do we have between PRO, GLY and general clusters.

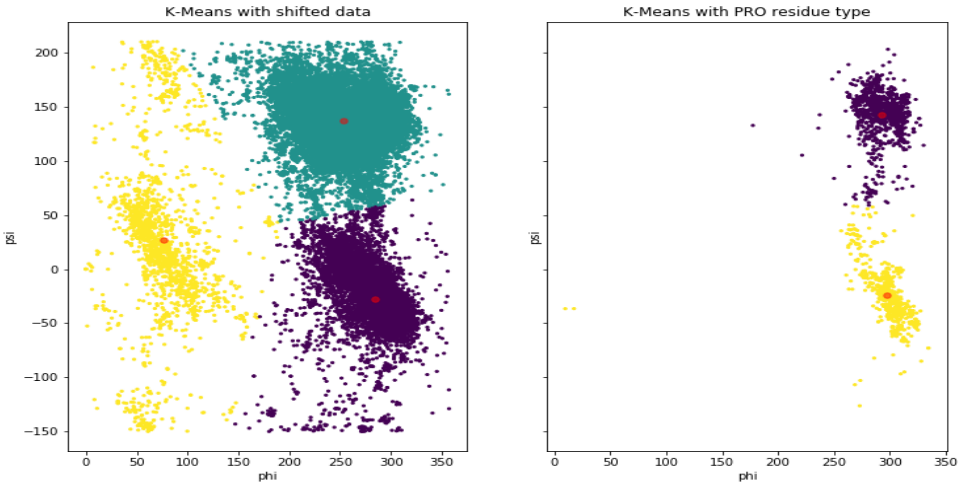

Figure 19: subplots to show difference between PRO type clusters and original data clusters

As we can see from Figure 19, it is obvious to say that PRO residue type data points spread out into two clusters whose core cluster points are somehow the same like in shifted data clusters.

Moreover, it should be pointed out that, GLY residue type is most common type among the other residue types. As can be seen from the figure 20 below, we can say that cluster of GLY residue types are similar to clusters of shifted data and GLY residue type composes main part of the data.

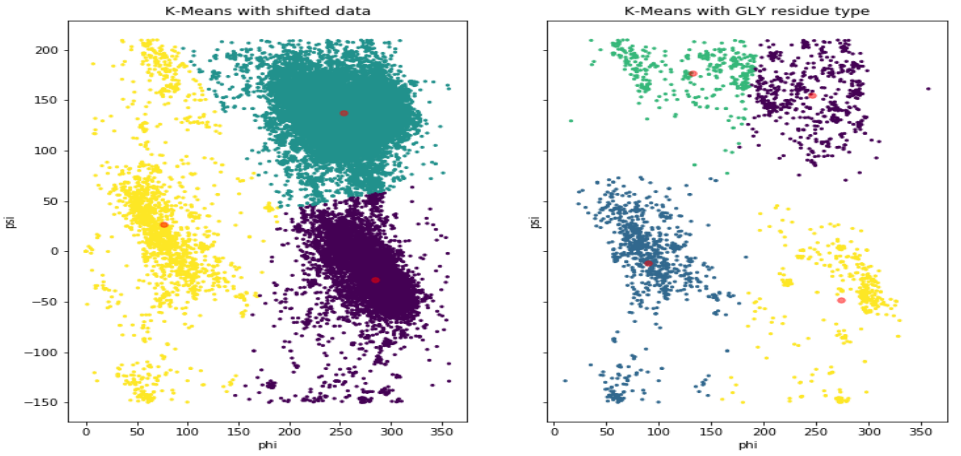

Figure 20: subplots to show difference between GLY type clusters and original data clusters