AI Assignment 3

Gille Gustav(10h), Westerberg Jacob (8h)

February 2 2023

1 Plotting ϕ and ψ distributions

1.1 A

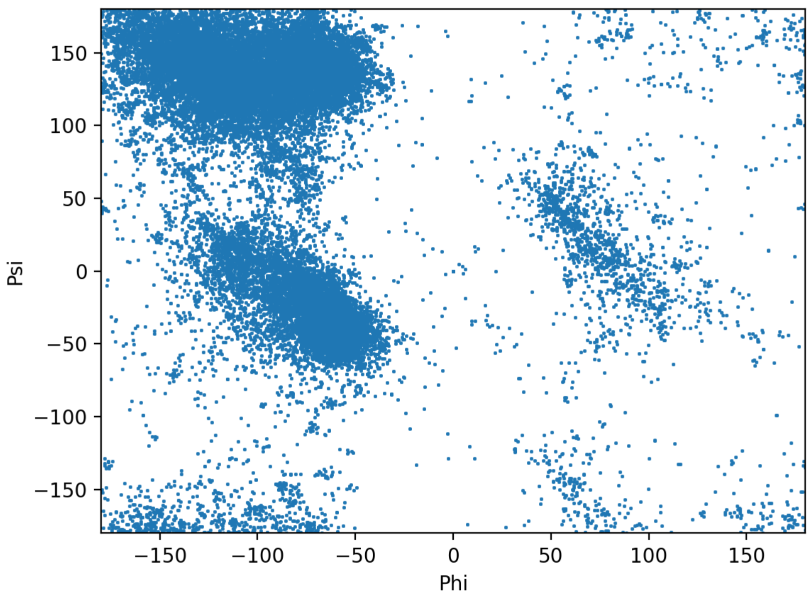

Figure 1: Scatter plot where ϕ on the y axis and ψ on the x axis

Following method was used to create scatterplot in figure 1:

```
def regularScatterPLot(x_values, y_values):
    plt.scatter(x_values, y_values, marker="o", s=1)
```

1.2 B

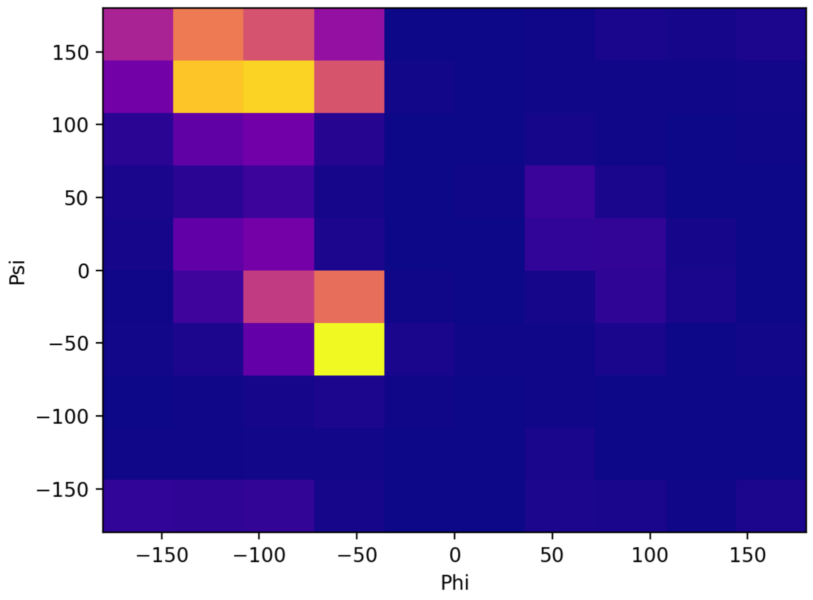

Figure 2: 2D histogram where ϕ on the y axis and ψ on the x axis. Bins value set to 10, see code below.

Here three distinct groups are seen, two very strong groups and one lesser to the right. Following code segment was used:

```
def create2DHistogram(x_values, y_values):
plt.hist2d(x_values, y_values, bins=10, alpha=1, cmap='plasma')
```

Using a higher bins value creates a higher resolution histogram, but the clearer grouping as seen in figure 2 is lost:

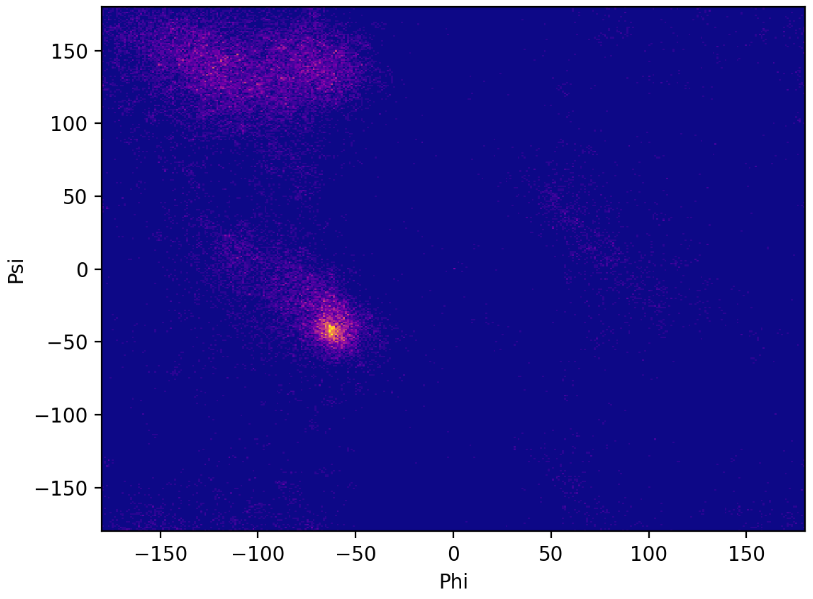

Figure 3: 2D histogram where ϕ on the y axis and ψ on the x axis. Bins value set to 350

2 K-means clustering ϕ and ψ distributions

At a glance it is obvious that three distinct major clusters are present in figure 1. Therefore a value of k = 3 seems resonable. A statement based on sheer human observation of a graph is seldom a sufficient scientific tool to make a valid scientific statement. As a consequence of this the elbow method is implemented to either prove or disprove the visual statement.

2.1 Applying the Elbow Method

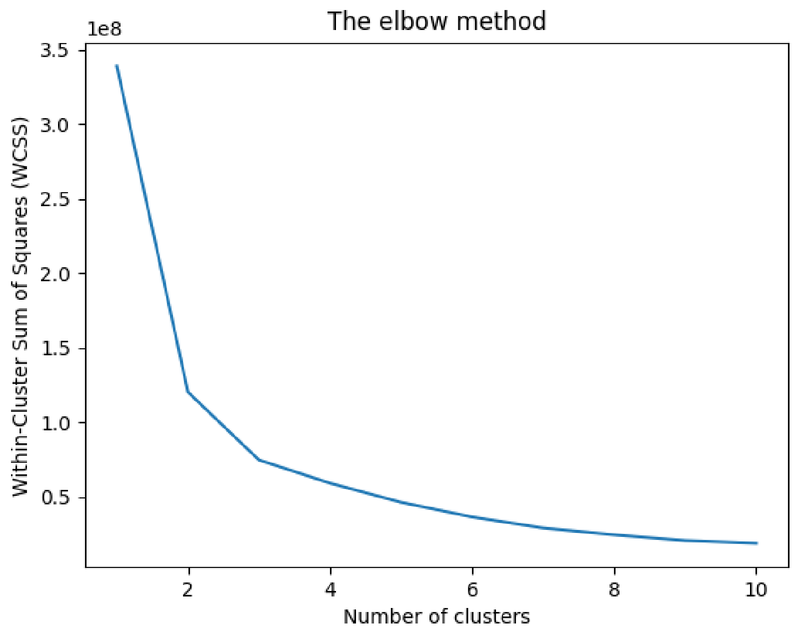

Figure 4: *This figure displays the relationship between WCSS and the number of clusters*

This graph is written by the following code:

```
def elbowMethod(data):
    #Creating empty list to store Within-Cluster Sum of Squares (
    WCSS) values
    WCSS = []

    #Creating for loop that runs kmeans with different numbers of
    clusters 1-11
    for i in range(1, 11):
        kmeans = KMeans(n_clusters=i, init='k-means++')
        kmeans.fit(data)
        #Appending the inertia value to the WCSS list
        WCSS.append(kmeans.inertia_)

    #Plotting the elbow graph
    plt.plot(range(1, 11), WCSS)
    plt.title('The elbow method')
    plt.xlabel('Number of clusters')
    plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
    plt.show()
```

The graph is the foundation to determine whether or not the graph has an "elbow" or not. The word elbow derives from the uncanny resemblance in the graph of a arm with and elbow. In our graph this happens between two and three, see figure 4. In mathematical terms this is the moment whereas the relative derivatives maximually differ. This doesn't have to be calculated since the graph already displays this.

It should be noted that the elbow method is not infallible. It is impractical when dealing with a very large number of clusters. Furthermore it is also not suitable when dealing with clusters that are really different in size/weird shapes and the borders between the clusters are really hard to differentiate. However this is not the case in this graph, see figure 1.

3 DBSCAN with optimized "MinPts" and ψ

3.1 Choosing "MinPts" and ϵ values

For choosing a good "MinPts" domain specific knowledge is applied. By analyzing the graph with the knowledge of three large clusters, a good "MinPts" should be values which classify those specific areas while also seperating the outliers as noise.
The graph below shows the elbow drop-off, with a dotted line at the specific point.

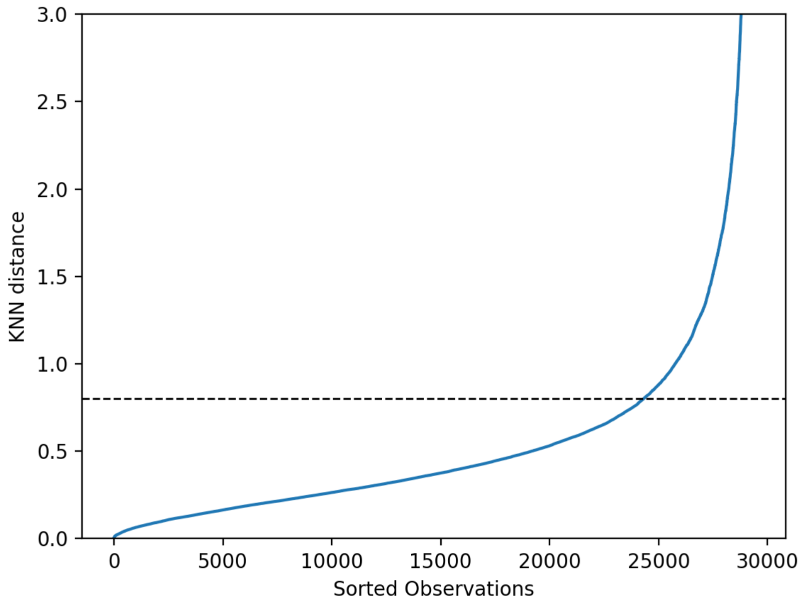

Figure 5: *The elbow graph*

The following code was used for graphing the elbow method in figure:

```
def calculateOptimizedEpsilon(dataset, neighbours):
    neighbors = NearestNeighbors(n_neighbors=neighbours)
    neighbors_fit = neighbors.fit(dataset)
    distances, indices = neighbors_fit.kneighbors(dataset)

    distances = np.sort(distances, axis=0)
    distances = distances[:, 1]
    plt.axhline(y=0.8, linewidth=1, linestyle='dashed', color='k')
    plt.plot(distances)
```

Listing 1: The elbow can be estimated to 0.8

The parameter ”neighbours” did not affect the ϵ value much more than the chosen one.
After trying multiple values of MinPts it never worked. Instead, through trial and error, an ϵ value of 13 with a MinPts set to 110 provided the most accurate data.
A reflection of this could be that the epsilon value might have been correlated ”in a positive way” against the k-means elbow value, but this does not mean that they are both a good measurement for each other, probably as density is more important in the DBSCAN than in the K-means algorithm.

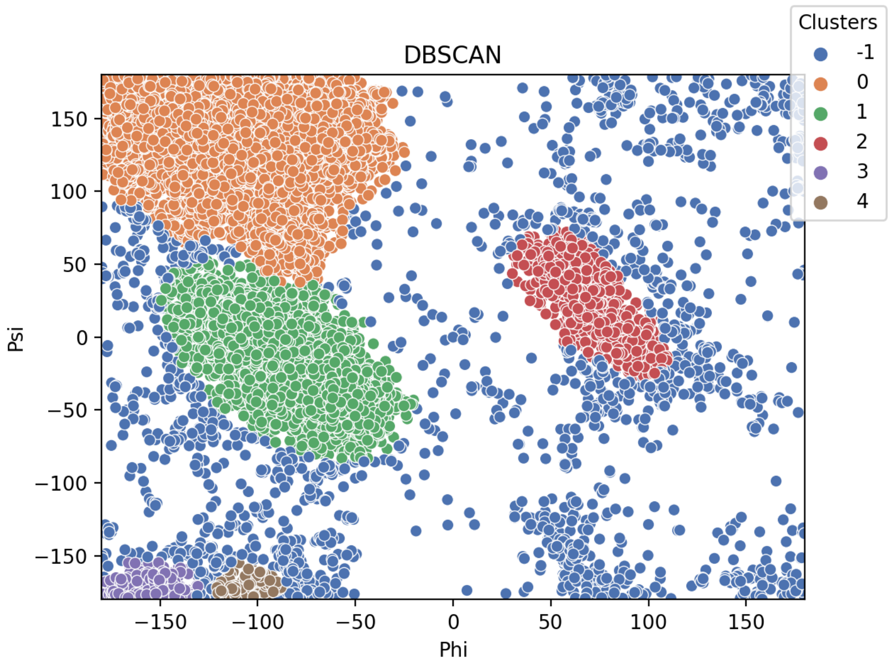

Figure 6: *the -1 cluster represents the outliers*

The follow code was used when creating this plot:

```
def createDBSCAN(epsilon, minsamples, dataFramed):
    return DBSCAN(eps=epsilon, min_samples=minsamples).fit(
    dataFramed)


if __name__ == '__main__':
    X = createPandaNPArray(readData(), "phi", "psi")
    dbscan = createDBSCAN(13, 110, X)
    snsPlot = sns.scatterplot(x=X[:, 0], y=X[:, 1], hue= dbscan.
    labels_, legend="full", palette="deep")
    sns.move_legend(snsPlot, "upper right", bbox_to_anchor=(1.13,
    1.15), title='Clusters')
```

Listing 2: function and main used for DBSCAN

In the question it is known that three distinct groups are searched for. By applying the knowledge that the graph is suppose to "wrap around" cluster 3 and 4 are easily associated with cluster 0. Thus this gives a more accurate plot than having two groups or not having a clear divide between cluster 0 and 1, which was the case otherwise. It is not known why the predicted ϵ value failed other than it was a bad prediction. By using:

```
from collections import Counter
...

 print(Counter(dbscan.labels_))
```

Listing 3: Counting clusters

The following outliers were counted to be 1996 (-1: 1996). The outliers were then against their amino acid type. It was found that roughly half of the outliers were represented by the amino acid residue type GLY. The following function was used, which produces figure 7:

```
def plot_outliers(data):
    data['acids'] = list(dbscan.labels_)
    outliers = data[data['acids'] == -1]
    outliers['residue name'].value_counts().sort_values().plot.bar
    ()
```

Listing 4: Creates a new row which is then used to unionize the outliers.

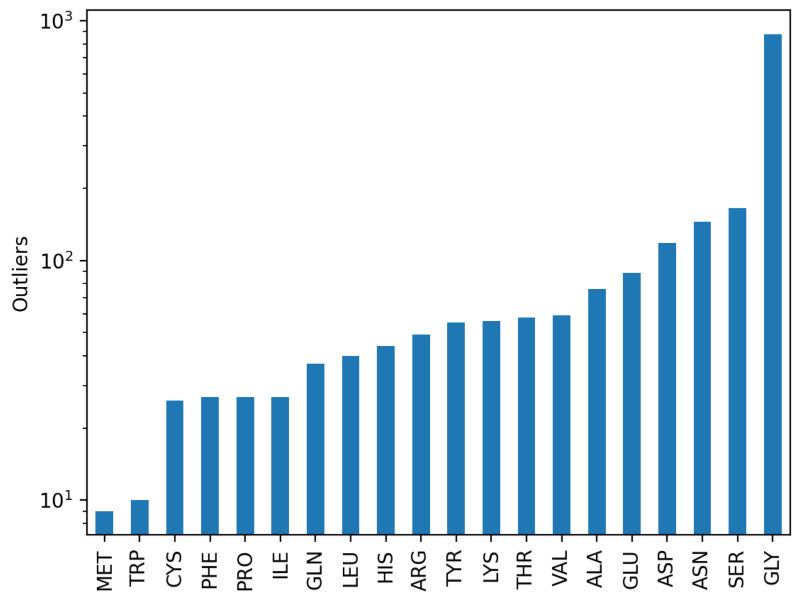

Figure 7: *The amino acids outlying frequency, y-axis given in* log_2

4 Stratifying amino acid residue type PRO

The follow code produces figure 8:

```
X = readData()
X = X.loc[X["residue name"] == "PRO"]
X = createPandaNPArray(X, "phi", "psi")

dbscan = createDBSCAN(13, 110, X)
snsPlot = sns.scatterplot(x=X[:, 0], y=X[:, 1], hue= dbscan.
labels_, legend="full", palette="deep")
sns.move_legend(snsPlot, "upper right", bbox_to_anchor=(1.13,
1.15), title='Clusters')

plt.show()
```

Listing 5: Plotting with the same DBSCAN inputs as earlier and created subset with only pro amino acid positions.

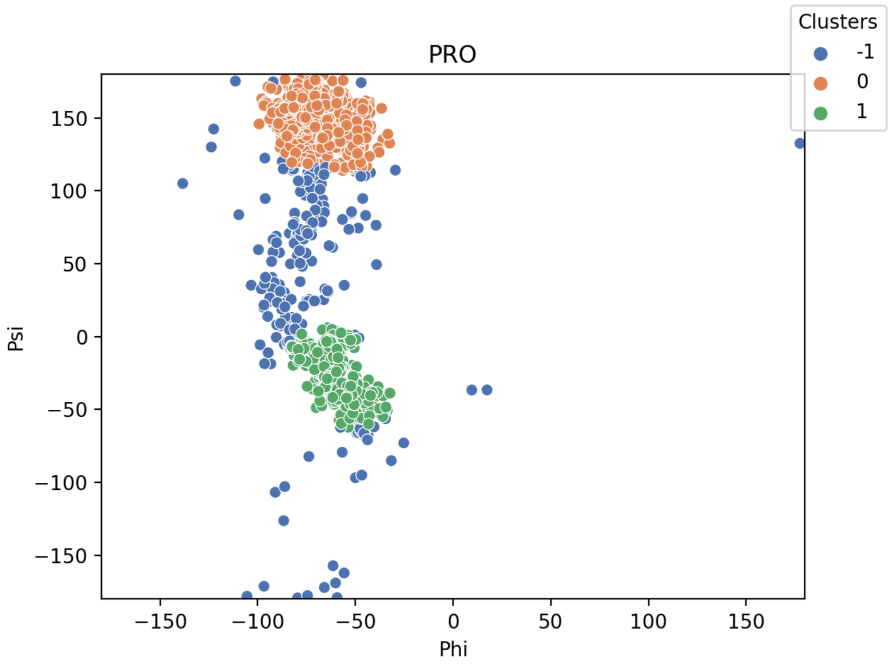

Figure 8: *The DBSCAN of the PRO amino acid.*

It is observed that the majority of PRO structures can be found in what is represented in both figure 6 and this figure as clusters 0 and 1. It is also seen that the angel of ϕ is fixed at an angle of $-100 \leq \phi \leq -20$ degrees while ψ revovles around $-75 \leq \psi \leq 0 \cap 0 \leq \psi \leq 190$ degrees (overlapping).