Intro to Machine Learning - Assignment 1

Maxmillan Ries s3118134, Cristian Rosiu s3742377

February 2020

Contents

1 PCA **1**
1.1 Data Exploration 1
1.2 Data Analysis: PCA 2
1.3 Dimensionality Reduction Evaluation 4

2 Hierarchical Clustering **8**
2.1 Data exploration 8
2.2 Data Analysis: Clustering 10
2.3 Clustering Evaluation 12

3 Individual Work **17**
A Assignment 1 Code 17

1 PCA

1.1 Data Exploration

a The Iris data set provided by contains samples from 3 distinct types of iris plants. The kind of data recorded consists of a list of 50 measurements per plant, for 4 different properties. These properties are the sepal length, the sepal width, the petal length and the petal width.
The data is presented in form of 5 columns, 150 rows, with the first 4 columns matching the previously described properties, and the 5th column being the kind of the plant the results stand for.

Along with the measurements, a table is provided in the "iris.names" file, which contains the min value per category, the max value, the mean, standard deviation and class correlation.

b In broader terms, data variation, is a measurement of the spread of values inside a data set. Specifically, the variance measures the distance of each value in a set from the mean, allowing it to be used to find the distance of each value from each other (Source: Slides and Recording).

As explained in part a of the assignment, the mean values for the data set were given. These values were rounded to the second decimal point. To ensure our results were correct, we computed it after calculating the mean ourselves, comparing it to the numbers calculated using the given mean. In our case, the following variance values were found. Note that the V_ours stands for the values obtained using our mean calculation:

Name	V_ours	V_given
Sepal Length	0.6889	0.6811
Sepal Width	0.1849	0.1868
Peal Length	3.0924	3.0976
Petal Width	0.5785	0.5776

As we can generally observed, the values stand to be similar when calculating, or using the given mean. Specifically to the variance however, it can be seen that the Petal length has a very large variance, showing that it's data stands to be far apart from the mean. The sepal width shows itself to have the lowest variance, implying that most values of this data set are stuck close each other, and the mean.

1.2 Data Analysis: PCA

a As shown in the appendix the PCA algorithm driven from the pseudocode in the slides was implemented. Before evaluation in more detail the procedure, we would like to precise that some functions were taken from the Numpy library, namely the linear algebra functions. The reason for this choice to avoid the hassle of implementing such well established functions as to focus on the core of the assignment and algorithm instead.

The first steps of our procedure was to the take the iris data set and convert it in to a Numpy array as to facilitate it's manipulation. To do this, the as_matrix function was implemented, which reads the lines of the data file and inserts them into a Numpy array. The type of flower is not included in the Numpy array, as it is not needed for the PCA calculations.

The PCA algorithm begins by calculating the mean of each category, something which we chose to do for higher accuracy (instead of taking the mean given, as mentioned in part 1) and immediately centering the data. After some testing with plug-and-play, we found that centering the data was most important, as it ensures that the first principal component (PC) describes the direction of Maximum Variance. If the subtraction was not done, the first PC might have corresponded to the mean of the data instead.

Following this, the algorithm computes the covariance matrix before using it to calculate both the eigenvalues and eigenvectors. As there are no eigenvalues or vectors for a non-square matrix, the covariance matrix is necessary to calculate.

Finally, using f_r (as described in the pseudocode), the number of components for reduction is chosen. Unlike other algorithms, where the number of components is an input, our algorithm uses the degree to which the variance should be preserved as an input, and calculates the minimum number of components required for it. The main idea of the algorithm is to keep the highest amount of variance while using the minimum amount of data, hence why the minimum is found.
Using this minimum number, the subset of eigenvectors is taken and the reduced dimensionality data is calculated. By default, we have chosen to use 0.95 (95%) for the variance, resulting in a number of components equal to 2 for this data set.

As we can see from the graph below, which is colored coded with red being the Iris Setosa, green being Iris Versicolor and blue being Virginia, the data offers the same conclusion as the given information, that one of the clusters is separable from the two.
The Iris Setosa's data is clustered closer to the -2 values of the first PC, while the other two iris species are clustered more towards the 2 values.

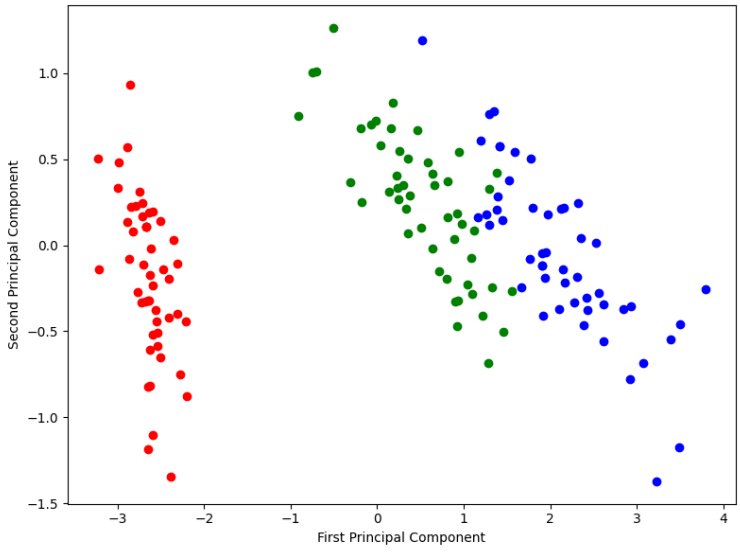

Figure 1: PCA implemented by Maxmillan and Cristi

In this figure, the blue dots correspond to the data of the Iris Virginia,

the green dots to the Iris Versicolor and the red dots to the Iris Setosa.

As we can see, 2 clutters are formed, composing of the iris Versicolor and Virginia. This graph clearly shows the separation of the iris Setosa from the two others, something which could not be differentiated using the 4 dimensional data.

b This graph was plotted using the PCA dimensionality reduction built into python. The data presented is flipped along the second principal component, but is otherwise identical with our results. We learned while trying to find if an error had occurred, that the eigenvalues are arbitarily negative, and that both graphs are equally correct.
As such, we can properly conclude that the Iris Setosa is linearly separable from the other type kinds of Iris plants.

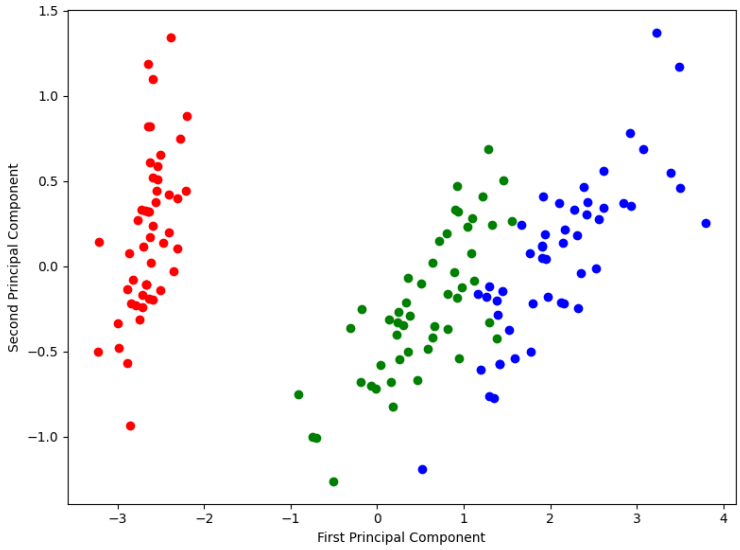

Figure 2: PCA from built in Python

1.3 Dimensionality Reduction Evaluation

a What we can expect from reducing the dimensionality, is a decrease in the average euclidean distance within a single class. In a perfect, ideal transformation, the average euclidean distance between all points would be unchanged, though in reality, a small reduction is expected, the difference being proportional to the number of dimensions reduced.
As we can see from the data we obtained, this expectation was correct. From 4 PC (original data), to 1 PC, the average inner class euclidean distance is continuously reduced, with a large proportion of the information

being lost when reducing down to 1 PC.

Plants	Average Euclidean Distances 1 PC
Setosa	0.244
Versicolor	0.680
Virginica	0.801

Plants	Average Euclidean Distances 2 PC
Setosa	0.621
Versicolor	0.886
Virginica	1.012

Plants	Average Euclidean Distances 3 PC
Setosa	0.673
Versicolor	0.977
Virginica	1.133

Plants	Average Euclidean Distances 4 PC
Setosa	0.698
Versicolor	0.997
Virginica	1.176

When comparing the data sets to one and other, we observed that the average inner euclidean distance between one set and another is larger than the distance between the members of a set. This of course makes sense. As the sets of data belong to different cluster, the average distance between both clusters is higher.
To calculate this difference, we chose to take the average euclidean distance of each pair of 1 class relative to another, and take the mean of those 50 values, as we felt that was the most concise, yet still effective way of presenting the information.

Plants	Average Euclidean Distances 1 PC
Setosa Vs Versicolor	3.172
Setosa Vs Virginica	4.749
Versicolor Vs Virginica	1.591

Plants	Average Euclidean Distances 2 PC
Setosa Vs Versicolor	3.276
Setosa Vs Virginica	4.801
Versicolor Vs Virginica	1.740

Plants	Average Euclidean Distances 3 PC
Setosa Vs Versicolor	3.293
Setosa Vs Virginica	4.816
Versicolor Vs Virginica	1.822

Plants	Average Euclidean Distances 4 PC
Setosa Vs Versicolor	3.298
Setosa Vs Virginica	4.822
Versicolor Vs Virginica	1.842

As we can see the pattern remains the same, whether comparing the data within a class or between two separate classes.
One important note about Dimensionality Reduction, is that the long pair euclidean distance is preserved more than the smaller, closer pairs. The reasons for this is that the outliers in the data remain further apart from one and other, and reducing the dimension of the data does not remove too much of the euclidean distance. For small values which might have large distance along a collapsed axis, the distance would change from significant to otherwise.

For the different level of reduction, the following alpha values were used.

- 1 Principle Component. Alpha = 0.92.
- 2 Principal Components. Alpha = 0.93.
- 3 Principle components. Alpha = 0.98.
- 4 Principle Components. Alpha = 1.0.

We can generally see that a majority of the information can be obtained using a single principle component. However, when comparing the average euclidean distance, the values seem significantly more distant from the initial values (4 PC's).

b As we have discussed before, the Iris Setosa seems to be separable from the other two plant species, as the data is clustered more towards the -2 values along the first Principal Component. An interesting observation between the data obtained in part e is that the Iris Setosa's average euclidean distance is smaller than that of the other two plants, showing that the data is less spread/scattered. The data seems to be more packed towards one and other, which means that the data is distributed closer to the mean than the data of the other two plants.

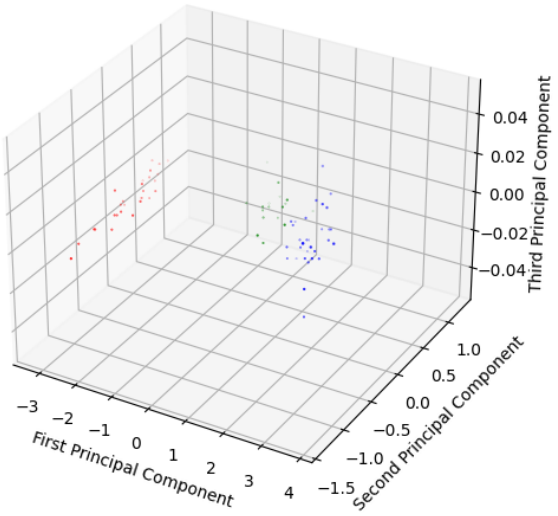

Figure 3: PCA 3D implemented by Maxmillan and Cristi

Looking at the 3 dimensional graph, the same conclusions can be made. Overall, the data follows to some extent what we were expecting. Should the data of each plant have varied too much, it would have been difficult to find a pattern between the plants. But overall, the data competently shows the separable plant from the 3.

The two dimensional graph + analysis, can be found in the Data Analysis: PCA part.

c Applying the same procedure to d, we can see that the results are nearly identical. Due to the arbitrary choice of +/- for a PC, the graph is flipped along the x and y axis, though the results otherwise show exactly the same as our own procedure (2D graph can be found in the section 2 above).

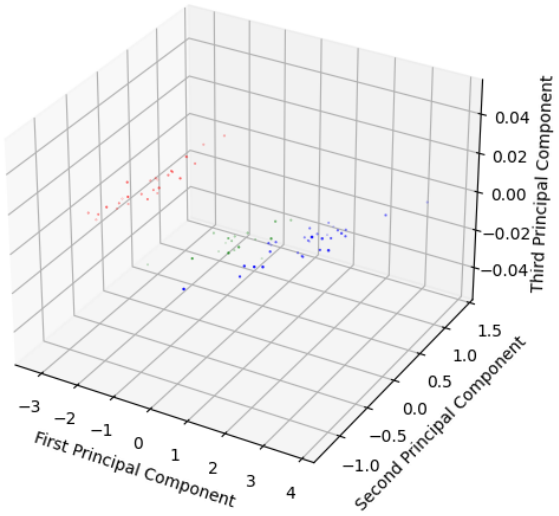

Figure 4: PCA 3D using built in Python

2 Hierarchical Clustering

2.1 Data exploration

a The data set is given has a 7606x2x68 data frame along with a 7606x1 labeling scheme. The idea of the data consists of 7606 faces, which are represented by the vectors of 2x68, which we modified to look like 1x136 by appending the two arrays to one and other appropriately.
The 7606x1 labeling scheme directly matches with the data set, as it provides the true class of each element of the data. This true class can be visualized as the "source" for simplicity sake. With this, the goal of the exercises to come are to cluster the raw data such that it matches the properties established using the labeling scheme, effectively performing some kind of supervised learning.

Each of the 68 data points consists of the data points representing a face, as described in more detail in the assignment guidelines,
Below is a bar plot of the data. There are several larger spikes on the graph, notably around the values of 300, 180 and 850, showing that large portion of the data belong to three specific true classes.
From this we can expect that the clustering of the data will result in 1 or 2 clusters containing a majority of the data, as they all come from the same "source".

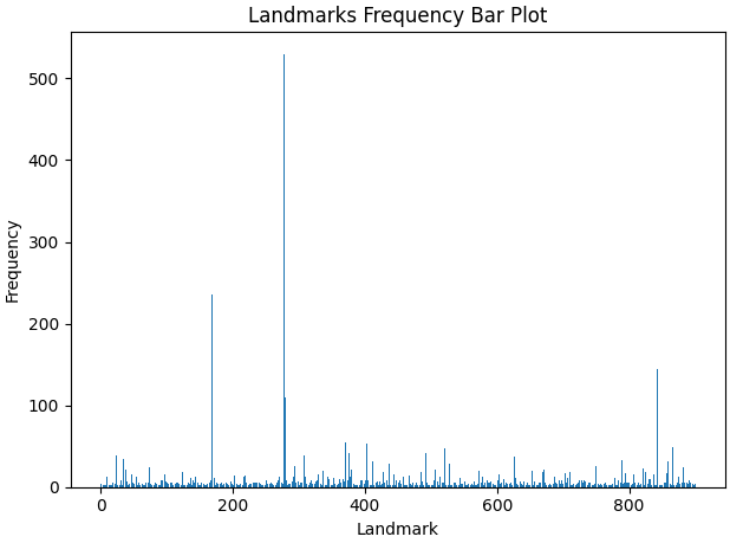

Figure 5: Landmark Frequency Barplot

b After applying the built-in PCA algorithm on the data and plot it, we got the following image:

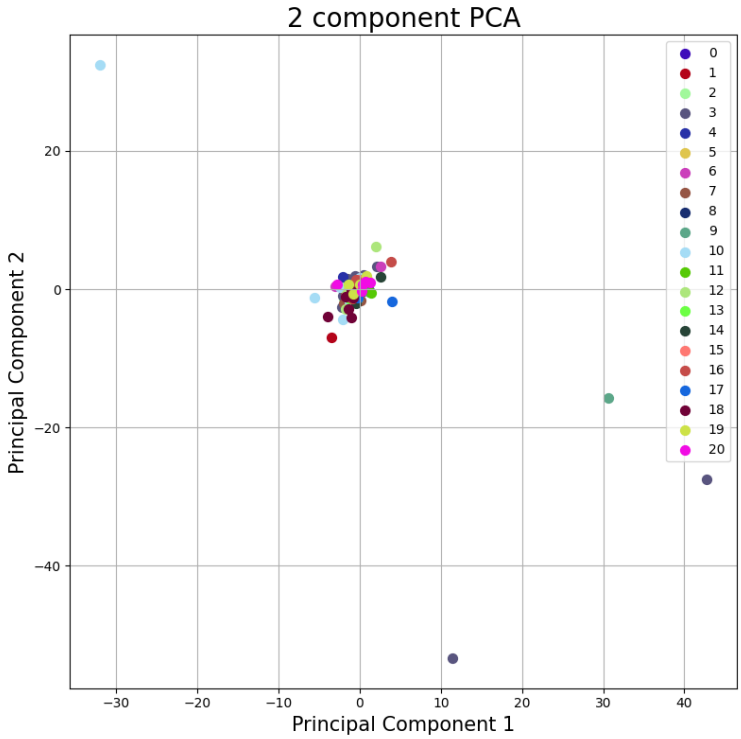

Figure 6: PCA Plot

The data was colored using the ground-given truth, which is the "source" or the labels provided along with the raw data.

From what we can observe, a large majority of the data is clustered so closely together that it is difficult to see the individual dots on the graph. As we center the data when performing PCA, we can see that most of the data is clustered around [0,0], with a few outliars belonging to classes 3, 9, and 10.

As per the observations which can be done using PCA, we can properly conclude that the face values are not linearly separable, which is something expected as they all have values between 0-67, representing different facial features.

2.2 Data Analysis: Clustering

a What we initially to approach this question was to implement the pseudo code obtained on the slides. From this, several dendograms were retrieved,

upon which we discovered some pattern.

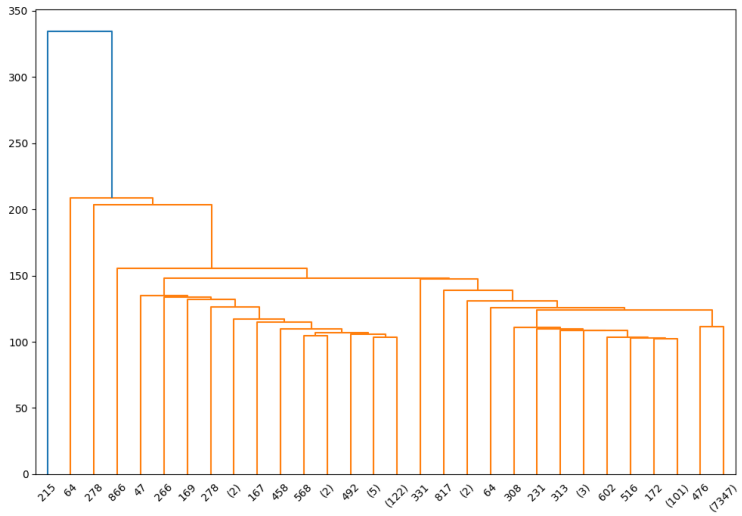

Figure 7: Example of an obtained dendogram

The idea we found behind the use of the dendogram, was that a horizontal "cut" could made at any point in the graph, which would show the number of clusters present with such an arbitrary choice. Following this procedure, we arbitrarily picked the number of clusters to evaluate to be 3, and continued with that number for part of the assignment.

However, after more research and study, we found that the AgglomerativeClustering function suited the goal of this assignment better. Using this function, as well as the k-means clustering methods built into some python libraries, we found with the "elbow method" a way to decide upon the best number of clusters for this particular data set.

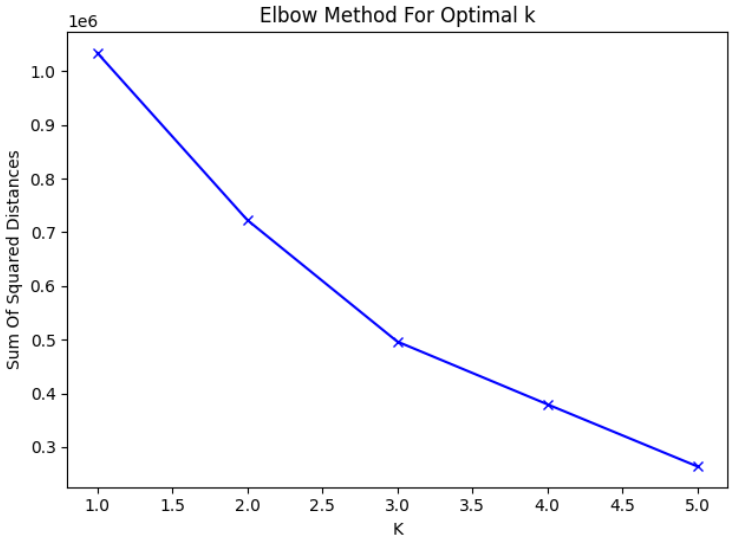

Figure 8: WSS vs K Graph

In our clustering analysis, we do the following:

- For each linkage method that exists, we create a model using the AgglomerativeClustering algorithm.
- We then fit our given data to the model and obtain the labels which match and cluster our data, similarly to how the data was provided.

We repeat these two processes for several values of k, where k is a number of clusters we arbitrarily decide.
Using the elbow method, we found that the value of 3 we had arbitrarily chose indeed fit the data best.

2.3 Clustering Evaluation

a To obtain the WSS and BSS values, we built our code on top of our AgglomerativeClustering analysis. The function compute_internal_measure() will compute either WSS or SSE for a specific model by using the formulas in the slides. One caveat here is that, instead of computing the means manually, we are using the already calculated centroids values of each cluster in the formula.

```
# center - centroid specific to current cluster
# data_center - center of all data
if measure == 'wss':
    sum = sum + (data[i, j] - center[j]) ** 2
```

```
elif measure == 'sse':
    sum = sum + (data[i,j] - data_center[j]) **2
```

To keep things simpler for ourselves in terms of implementation, we used the formula SSE - BSS - WSS = 0, to calculate the BSS using the WSS and the SSE. The results are shown further in this report.

b To get the TP, TN, FP and FN's, we used the ground-given truth. Taking all unique pairs obtainable from the ground given truth, we used our perf_measure function. The process after taking the pairs, was to compare if the points were appropriately clustered similarly.
If the values in the ground-given truth were from the same source, then a TP would mean that our clustering algorithm would cluster the points together. Same for the TN. The FP and FN's were obtained when our algorithm obtained the opposing result from what was expected.

One detail we did not have the time to fully understand and experiment with was the low level of accuracy this method caused. Despite that however, the method did seem to show some good results.

c Our results:

```
FOR K = 5

Linkage: SINGLE Affinity: CITYBLOCK Number of clusters: 5
WSS: 1010199.5054383497 BSS: 24216.494561914937
Accuracy 0.008596225402090357
Precision 0.008596420236737554
Recall 0.9973703544601599
F score 0.017045920231683113


Linkage: COMPLETE Affinity: CITYBLOCK Number of clusters: 5
WSS: 363219.9303738315 BSS: 671196.0696264331
Accuracy 0.008431921758857068
Precision 0.008437374235979301
Recall 0.9288148930276253
F score 0.016722837857315202


Linkage: AVERAGE Affinity: CITYBLOCK Number of clusters: 5
WSS: 405346.201198245 BSS: 629069.7988020196
Accuracy 0.008431952178929294
Precision 0.008437177116833674
Recall 0.9315810392519762
F score 0.016722897684291513
```

```
Linkage: WARD Affinity: EUCLIDEAN Number of clusters: 5
WSS: 274549.6143291996 BSS: 759866.385671065
Accuracy 0.00841681336449576
Precision 0.008423493196385147
Recall 0.9138961711557995
F score 0.016693123821317078

--------------------------------------------------------------------------------

FOR K = 3

Linkage: SINGLE Affinity: CITYBLOCK Number of clusters: 3
WSS: 1023269.752598043 BSS: 11146.247402221663
Accuracy 0.008611832025335293
Precision 0.008611855256918476
Recall 0.9996868513708282
F score 0.017076603212243448


Linkage: COMPLETE Affinity: CITYBLOCK Number of clusters: 3
WSS: 583846.4994598784 BSS: 450569.5005403862
Accuracy 0.008437818054999187
Precision 0.008443072038558133
Recall 0.9313160673349847
F score 0.01673443400064752


Linkage: AVERAGE Affinity: CITYBLOCK Number of clusters: 3
WSS: 817103.667959189 BSS: 217312.33204107557
Accuracy 0.008620709580775621
Precision 0.008621592135858303
Recall 0.9882649558580875
F score 0.017094056266916714


Linkage: WARD Affinity: EUCLIDEAN Number of clusters: 3
WSS: 512658.4814312568 BSS: 521757.5185690078
Accuracy 0.008576083396267083
Precision 0.008580391482480897
Recall 0.9446931344170417
F score 0.017006319180974592
```

The best clustering number k was evaluated to be 3 by our algorithm.

Using the recall as the base measurement, along wit the cleanliness and clarity of the elbow graphs, we concluded that the best linkage method was "single". The following dendogram was obtained using what we consider to be our optimum parameters:

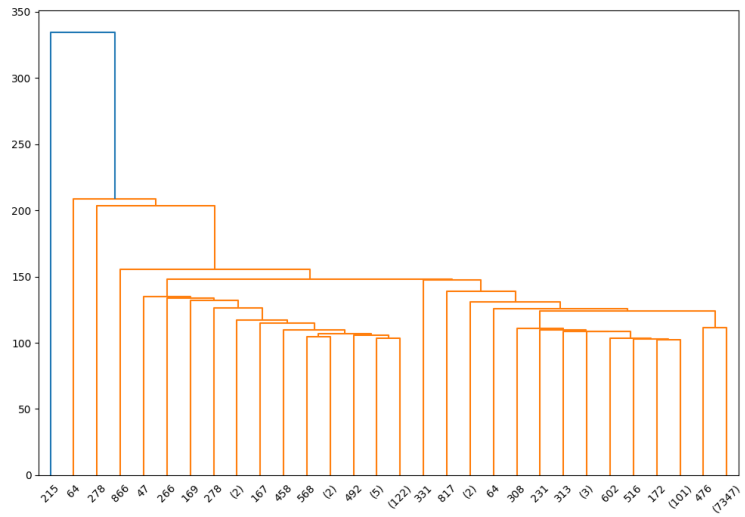

Figure 9: Example of an obtained dendogram

The dendogram we obtained directly confirmed our hypothesis to be true. One of the three obtained clusters, consisted of a single element, another one consisted of 141 elements and the last one of 7464 elements. As we predicted, a larger portion of the data was clustered together into single cluster due to the similarity of the data.

d We unfortunately did not have the time to evaluate much about the threshold and merge distance. Although did experiment a little bit with it, as a parameter in the AgglomerativeClustering function, we did not have time to form any conclusions.

e **Bonus** When running the process on the PCA reduced data set, we can properly visualize the correlation per data point, which is harder to observe on a dendogram (as students seeing it for the first time).
Below are several images showing processed data of part b used in the same process as part c.

```
K = 3 WITH REDUCED DATA

Linkage: SINGLE Affinity: CITYBLOCK Number of clusters: 3
WSS: 489724982.3983341 BSS: 206411.35947698355
```

```
Accuracy 0.008614308711375144
Precision 0.00861431079686041
Recall 0.9999718969178948
F score 0.017081472346710903


Linkage: COMPLETE Affinity: CITYBLOCK Number of clusters: 3
WSS: 63921677.62978186 BSS: 426009716.1280292
Accuracy 0.024843246741248035
Precision 0.02484479689101804
Recall 0.9974948109666256
F score 0.048482042146920534


Linkage: AVERAGE Affinity: CITYBLOCK Number of clusters: 3
WSS: 133124219.02163959 BSS: 356807174.7361715
Accuracy 0.015454554660580246
Precision 0.015454865751933248
Recall 0.9986992287711325
F score 0.030438692878276552

Linkage: WARD Affinity: EUCLIDEAN Number of clusters: 3
WSS: 75482345.13391116 BSS: 414449048.62389994
Accuracy 0.021415169524897876
Precision 0.0214152984453047
Recall 0.9997189691789484
F score 0.04193235065200559
```

Comparing it to the raw data analysis, we can see that the accuracy and recall increased. The recall very similar for every kind of linkage, unlike the raw data, which had a little variation to it.
The WSS and BSS values are significantly larger and we can now see which linkage method would be best as the data could be visualized on a 2D scatter plot, which is colored coded according to the obtained labels from the clustering method.

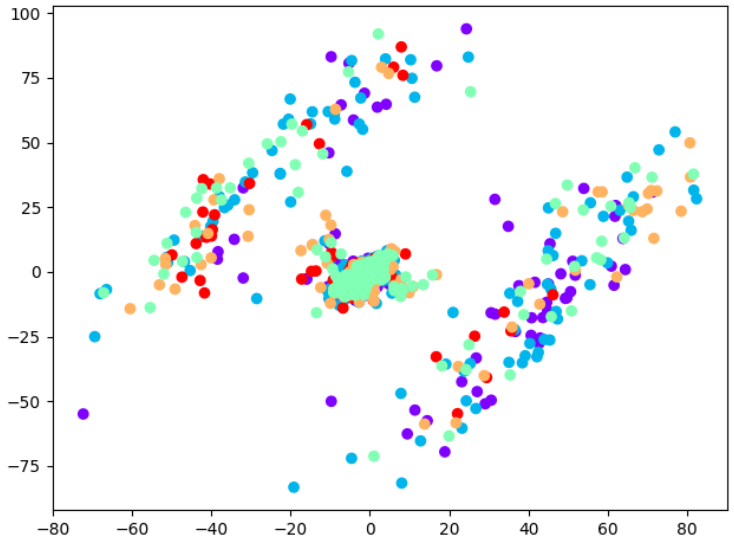

Figure 10: PCA Bonus

From a visual standpoint, it seems the clustering algorithm has not worked too well in this case.

3 Individual Work

Throughout this assignment, both members of the group did equal amounts of work. Both individuals worked on the code, and on the documents. There were no issues when working together, everything went smoothly.
In full honesty, everything was worked on together as part of the learning process, a proper 50/50. Github was used as a means of sharing the code between one and other.

Appendix

A Assignment 1 Code

```
1 import os
2 import numpy
3 import matplotlib.pyplot as plt
4 import math
5 from itertools import combinations
6 from itertools import product
7 from sklearn.decomposition import PCA
```

```
8
9
10  # Compute variance per plant feature.
11  def compute_features_variance(sample_size, data_mean):
12      fileDir = os.path.dirname(os.path.relpath('__file__'))
13      filename = os.path.join(fileDir, 'iris-data-set/iris.data')
14
15      data_file = open(filename, 'r')
16      results = [0 for i in range(4)]
17
18      for line in data_file.readlines():
19          line = line.split(",")
20          line.pop()
21
22          results[0] += (float(line[0]) - data_mean[0]) ** 2
23          results[1] += (float(line[1]) - data_mean[1]) ** 2
24          results[2] += (float(line[2]) - data_mean[2]) ** 2
25          results[3] += (float(line[3]) - data_mean[3]) ** 2
26
27      # Calculate Variance
28      results = [round(result / sample_size, 4) for result in results]
29      data_file.close()
30
31      return results
32
33
34  def compute_variance_ratio(eigenvalues, alpha):
35      eigenvalue_sum = numpy.sum(eigenvalues)
36
37      ratio = 0
38      # Dimensionality number.
39      r = 0
40      while ratio < alpha:
41          ratio = ratio + (eigenvalues.pop(0) / eigenvalue_sum)
42          r = r + 1
43      return ratio, r
44
45
46  def pca(data, alpha):
47      # Get mean of values and center data accordingly
48      mean = data.mean(axis=0)
49      center_data = data - mean
50
51      # Calculate the covariance
52      covariance = numpy.dot(numpy.transpose(center_data), center_data) / 150
53
54      # Get the Eigenvalues & Eigenvectors
55      eigenvalues, eigenvectors = numpy.linalg.eig(covariance)
```

```
56
57      eigenvalues = eigenvalues.tolist()
58
59      # Compute the ratio of total variance
60      variance_ratio, r = compute_variance_ratio(eigenvalues, alpha)
61
62      # Do dimensionality reduction
63      eigenvectors = numpy.transpose(eigenvectors)
64      eigenvectors = eigenvectors[0:r]
65
66      result = numpy.dot(center_data, numpy.transpose(eigenvectors))
67
68      return result
69
70
71  # Transform data frame to numpy array
72  def data_to_numpy():
73      file_dir = os.path.dirname(os.path.abspath('__file__'))
74      filename = os.path.join(file_dir, 'iris-data-set/iris.data')
75      data_file = open(filename, 'r')
76
77      matrix = []
78
79      for line in data_file.readlines():
80          line = line.split(",")
81          line.pop()  # Remove names from array.
82          matrix.append(line)
83
84      return numpy.array(matrix).astype(numpy.float)
85
86
87  def average_euclidean_distance(points):
88      esum = 0
89      for tuple in points:
90          esum = esum + math.sqrt(sum([(a - b) ** 2 for a, b in zip(tuple[0], tuple[1])]))
91      return esum / len(points)
92
93
94  def plot_pca(array, dimension):
95      plt.figure()
96
97      if dimension == 2:  # 2D Plot
98          plt.scatter(array[:50, 0], array[:50, 1], color='red')
99          plt.scatter(array[50:100, 0], array[50:100, 1], color='green')
100         plt.scatter(array[100:150, 0], array[100:150, 1], color='blue')
101
102         # labeling x and y axes
103         plt.xlabel('First Principal Component')
```

```
104             plt.ylabel('Second Principal Component')
105
106         elif dimension == 3:  # 3D plot
107             fig = plt.figure()
108             ax = fig.add_subplot(111, projection='3d')
109             plt.scatter(array[:50, 0], array[:50, 1], array[:50, 2], color='red', marker='o')
110             plt.scatter(array[50:100, 0], array[50:100, 1], array[50:100, 2], color='green', marker='o')
111             plt.scatter(array[100:150, 0], array[100:150, 1], array[100:150, 2], color='blue', marker='o')
112
113             # labeling x, y and z axes
114             ax.set_xlabel('First Principal Component')
115             ax.set_ylabel('Second Principal Component')
116             ax.set_zlabel('Third Principal Component')
117
118         plt.show()
119
120
121     # ---------------------------------------- Data Exploration ----------------------------------------
122     print("------ Data Exploration ------")
123     print("Variance Per Feature: ")
124     variance_array = compute_features_variance(sample_size=150, data_mean=numpy.array([5.84, 3.05, 3.76, 1.20]))
125     print("Sepal Length Variance: ", variance_array[0])
126     print("Sepal Width Variance: ", variance_array[1])
127     print("Petal Length Variance: ", variance_array[2])
128     print("Petal Width Variance: ", variance_array[3], '\n')
129
130     # ---------------------------------------- Data Analysis: PCA ----------------------------------------
131
132     data = data_to_numpy()
133
134     # Compute PCA for 3 different dimensions using our algorithm.
135     PC_1 = pca(data, 0.92)
136     PC_2 = pca(data, 0.93)
137     PC_3 = pca(data, 0.98)
138     PC_4 = pca(data, 1)
139
140     # Compute PCA result using sklearn library.
141     pca_sklearn = PCA(n_components=2)
142     transformed_data = pca_sklearn.fit_transform(data)
143
144     # Plot both PCA results so we can compare them. PART D
145     plot_pca(PC_2, 2)
146     plot_pca(transformed_data, 2)
147
148     # ---------------------------------------- Dimensionality reduction evaluation ----------------------------------
149     print("------ Dimensionality reduction evaluation ------")
150
151     # Compute euclidean distance per class and print it.
```

```
# Euclidean distance for 1 PC
print("Average Euclidean Distances 1 PC:")
print(average_euclidean_distance(list(combinations(PC_1[:50], 2))))
print(average_euclidean_distance(list(combinations(PC_1[50:100], 2))))
print(average_euclidean_distance(list(combinations(PC_1[100:150], 2))), '\n')

# Euclidean distance for 2 PC
print("Average Euclidean Distances 2 PC:")
print(average_euclidean_distance(list(combinations(PC_2[:50], 2))))
print(average_euclidean_distance(list(combinations(PC_2[50:100], 2))))
print(average_euclidean_distance(list(combinations(PC_2[100:150], 2))), '\n')

# Euclidean distance for 3 PC
print("Average Euclidean Distances 3 PC:")
print(average_euclidean_distance(list(combinations(PC_3[:50], 2))))
print(average_euclidean_distance(list(combinations(PC_3[50:100], 2))))
print(average_euclidean_distance(list(combinations(PC_3[100:150], 2))), '\n')

# Euclidean distance for 4 PC
print("Average Euclidean Distances 4 PC:")
print(average_euclidean_distance(list(combinations(PC_4[:50], 2))))
print(average_euclidean_distance(list(combinations(PC_4[50:100], 2))))
print(average_euclidean_distance(list(combinations(PC_4[100:150], 2))), '\n')

# Euclidean distance between each individual 1 PC component
print("Average Euclidean Distances 1 PC between each different component")
print(average_euclidean_distance(list(product(PC_1[:50], PC_1[50:100]))))
print(average_euclidean_distance(list(product(PC_1[:50], PC_1[100:150]))))
print(average_euclidean_distance(list(product(PC_1[50:100], PC_1[100:150]))), '\n')

# Euclidean distance between each individual 2 PC component
print("Average Euclidean Distances 2 PC between each different component")
print(average_euclidean_distance(list(product(PC_2[:50], PC_2[50:100]))))
print(average_euclidean_distance(list(product(PC_2[:50], PC_2[100:150]))))
print(average_euclidean_distance(list(product(PC_2[50:100], PC_2[100:150]))), '\n')

# Euclidean distance between each individual 3 PC component
print("Average Euclidean Distances 3 PC between each different component")
print(average_euclidean_distance(list(product(PC_3[:50], PC_3[50:100]))))
print(average_euclidean_distance(list(product(PC_3[:50], PC_3[100:150]))))
print(average_euclidean_distance(list(product(PC_3[50:100], PC_3[100:150]))), '\n')

# Euclidean distance between each individual 4 PC component
print("Average Euclidean Distances 4 PC between each different component")
print(average_euclidean_distance(list(product(PC_4[:50], PC_4[50:100]))))
print(average_euclidean_distance(list(product(PC_4[:50], PC_4[100:150]))))
print(average_euclidean_distance(list(product(PC_4[50:100], PC_4[100:150]))), '\n')
```

```
# Compare 3D graphs of both algorithms
pca_sklearn = PCA(n_components=3)
transformed_data = pca_sklearn.fit_transform(data)

plot_pca(PC_3, 3)
plot_pca(transformed_data, 3)
\end{lstlisting}

\subsection{Assignment 2 Code}
\begin{minted}[
frame=lines,
framesep=1mm,
baselinestretch=1,
fontsize=\footnotesize,
linenos
]{python}
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import pandas as pd
import random
from sklearn.metrics import confusion_matrix
from sklearn.metrics import multilabel_confusion_matrix
from sklearn.cluster import AgglomerativeClustering, KMeans
from sklearn.neighbors.nearest_centroid import NearestCentroid
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from scipy.spatial.distance import pdist
from scipy.cluster.hierarchy import dendrogram, linkage
import scipy.cluster.hierarchy as shc
import sys


def plot_pca(data_frame,targets):
    fig = plt.figure(figsize=(8, 8))
    ax = fig.add_subplot(1, 1, 1)
    ax.set_xlabel('Principal Component 1', fontsize=15)
    ax.set_ylabel('Principal Component 2', fontsize=15)
    ax.set_title('2 component PCA', fontsize=20)

    colors = [random_color() for i in range(len(targets))]
    colors = set(colors)

    for target, color in zip(targets, colors):
        indicies_to_keep = data_frame['target'] == target
        ax.scatter(data_frame.loc[indicies_to_keep, 'principal component 1']
                   , data_frame.loc[indicies_to_keep, 'principal component 2']
```

```
                    , c=color
                    , s=50)
    ax.legend(targets)
    ax.grid()
    plt.show()


def random_color():
    r = lambda: random.randint(0, 255)
    return '#%02X%02X%02X' % (r(), r(), r())


def freq_bar(data, data_frequency, title, x_label, y_label):
    plt.bar(data, data_frequency)
    plt.title(title)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.show()


# Computes the list of sums of Squared Errors for all the points (WSS)
def elbow_method(data, k_max):
    # Compute WSS of each
    sse = []
    for k in range(1, k_max + 1):
        k_means = KMeans(n_clusters=k)
        k_means = k_means.fit(data)
        sse.append(k_means.inertia_)

    # Plot WSS vs K
    K = range(1, k_max + 1)
    plt.plot(K, sse, 'bx-')
    plt.xlabel('K')
    plt.ylabel('Sum Of Squared Distances')
    plt.title('Elbow Method For Optimal k')
    plt.show()


# Computes WSS/SSE for a specific number of clusters K
def compute_internal_measure(data, clusters_labels, centroids, measure):
    data_center = np.mean(data, axis=0)
    sum = 0

    for i in range(len(data)):
        # Get the list of means for the a specific data point (center of a cluster).
        center = centroids[clusters_labels[i]]
        for j in range(len(data[1])):
            # Compute either wss or bss for each data point and add it to the sum
```

```
            if measure == 'wss':
                sum = sum + (data[i, j] - center[j]) ** 2
            elif measure == 'bss':
                sum = sum + clusters_labels.count(clusters_labels[i]) * (data_center[j] - center[j]) ** 2
            elif measure == 'sse':
                sum = sum + (data[i,j] - data_center[j]) **2
            else:
                raise Exception("Wrong input for measure")
    # Final sum is the wss/bss value of a specific cluster.
    return sum

def perf_measure(true_labels, predicted_labels):
    TP = 0
    FN = 0
    FP = 0
    TN = 0
    for i in range(len(true_labels) - 1):
        for j in range(i, len(true_labels)):
            if true_labels[i] == true_labels[j] and predicted_labels[i] == predicted_labels[j]:
                TP += 1
            elif true_labels[i] == true_labels[j] and predicted_labels[i] != predicted_labels[j]:
                FN += 1
            elif true_labels[i] != true_labels[j] and predicted_labels[i] == predicted_labels[j]:
                FP += 1
            elif true_labels[i] != true_labels[j] and predicted_labels[i] != predicted_labels[j]:
                TN += 0

    return TP, FP, TN, FN

# Apply dimensionality reduction on a dataset
def d_reduction(data, n_comp, targets):
    pca = PCA(n_components=n_comp)
    # Apply PCA Algorithm in order to reduce the dimension to the desired number.
    principal_components = pca.fit_transform(data)
    # Convert result to pandas data frame
    principal_df = pd.DataFrame(data=principal_components
                                , columns=['principal component 1', 'principal component 2'])

    # Add targets column for better visualization of data
    final_df = pd.concat([principal_df, targets], axis=1)

    # Returns new data as a panda dataframe
    return final_df

def h_clustering_analysis(x, k):
    linkages = ['single', 'complete', 'average', 'ward']

    for link in linkages:
```

```
344             print("Linkage:", link.upper(), "Affinity:", 'cityblock' if link.lower() != 'WARD' else 'EUCLIDEAN',
345                   "Number of clusters:", k)
346             model = AgglomerativeClustering(n_clusters=k, affinity='euclidean', linkage=link) if link.lower() == 'war
347                                             else AgglomerativeClustering(n_clusters=k, affinity='cityblock', linkage=
348             pred_labels = model.fit_predict(x)
349             cluster_labels = model.labels_
350
351             clf = NearestCentroid()
352             clf.fit(x, pred_labels)
353
354             wss = compute_internal_measure(x, cluster_labels, clf.centroids_, 'wss')
355             sse = compute_internal_measure(x, cluster_labels, clf.centroids_, 'sse')
356             bss = sse - wss
357             print("WSS:", wss, "BSS:", bss)
358
359             # ----------- Part E -----------
360             TP, FP, TN, FN = perf_measure(landmark_data, cluster_labels)
361
362             accuracy = (TP + TN) / (TP + TN + FP + FN)
363             precision = TP / (TP + FP)
364             recall = TP / (TP + FN)
365             f_score = (2 * precision * recall) / (precision + recall)
366             print("Accuracy", accuracy)
367             print("Precision", precision)
368             print("Recall", recall)
369             print("F score", f_score)
370             print()
371             print("-----------------------------------------------")
372             print()
373
374
375 features_data = np.load("IML_lab2_clustering/features.npy")
376 landmark_data = np.load("IML_lab2_clustering/gt_facialLandmarks.npy")
377 landmark_df = pd.DataFrame(data=landmark_data, columns=['target'])
378
379 features_data = features_data.reshape(7606, 136)
380
381 # ---------------------- DATA EXPLORATION ----------------------
382 x = StandardScaler().fit_transform(features_data)
383
384 # Bar plot for better visualisation of the data
385 unique_labels = landmark_df['target'].unique().tolist()
386 labels_freq = [landmark_data.tolist().count(label) for label in unique_labels]
387 freq_bar(unique_labels, labels_freq, "Landmarks Frequency Bar Plot", "Landmark", "Frequency")
388
389 # Reduce data to 2 dimensions
390 reduced_data = d_reduction(x, 2, landmark_df)
391 reduced_data = reduced_data[reduced_data['target'] < 21]
```

```
# Plot newly reduced data
plot_pca(reduced_data)


# ----------- Part C + D -----------
# Try H Clustering with K = 5
h_clustering_analysis(x, 5)

# Apply elbow method to check which K is Optimal
elbow_method(x, 5)

# Try H Clustering with optimal K
h_clustering_analysis(x, 3)

# Bonus
h_clustering_analysis(reduced_data.to_numpy(), 3)

model = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')

reduced_clusters_labels = model.fit_predict(reduced_data.to_numpy())
plt.scatter(reduced_data['principal component 1'], reduced_data['principal component 2'], c = reduced_clusters_la
plt.show()
```