CSE 5243: Introduction to Data Mining

k-means clustering

CHANDRASEKAR SWAMINATHAN (.42)

Introduction

This report discusses the results of the implemented k-means clustering algorithm and an off-the-shelf clustering algorithm on the following three datasets:

- **Dataset 1: TwoDimEasy** - Two numeric independent variables; 2 true, relatively wellseparated clusters; 300 examples.
- **Dataset 2: TwoDimHard** - Two numeric independent variables; 4 true, slightly overlapping clusters; 400 examples.
- **Dataset 3: Wine Quality**

Part 1

The implemented k-means clustering algorithm, in each iteration, computes the distance between all points and all the centroids, it then associates each point to the closest centroid, thereby creating clusters of points. The centroids of these newly formed clusters are computed and are used for the subsequent iteration of the algorithm.

Design decisions

Distance measure

The implemented k-means clustering algorithm uses the standard euclidean distance.

Initial centroids

The algorithm picks the k random points from the entire dataset as the initial centroids. There is no guarantee in this approach that the each one of the initial centroids would belong to a different cluster. Choosing two points that are in the same cluster could result in very poor clusters being formed. Despite these limitations, this strategy was used to pick the initial centroids due to the fact it is very straightforward and easy to implement. To avoid poor choice of initial centroids the clustering algorithm was run 10 times for each value of k and the best of the 10 runs is taken as the output.

Handling Empty Clusters

When we encounter an empty cluster, we compute the distance between each point and the centroid of the cluster it belongs, and then choose the point that is farthest away from the centroid of its cluster as the new centroid and continue with the k-means clustering algorithm.

Stopping condition

The implemented algorithm stops only when the centroids do not change between two subsequent iterations of the algorithms. For the above three datasets, the algorithm has always converged quickly to the final centroids.

SSE, SSB, TSS and Silhouette Width

In this section, we look at both SSE (Sum of Squared Errors) and SSB (Sum of Squares between clusters) at the cluster level and at an overall level (sum of SSE/SSB values of all clusters). A lower SSE value indicates that the cluster is closely packed and is an indication how good the cluster is. The total SSE is computed as follows:

$$Total\ SSE = \sum_{i=1}^{K} \sum_{p \in C_i} distance(p,\ centroid(C_i))^2$$

$$Total\ SSB = \sum_{i=1}^{K} \left|C_i\right| distance\ (global\_centroid\ ,\ centroid(C_i))^2$$

$$K \quad - \ number\ of\ clusters$$

$$C_i \quad - \ Cluster\ i$$

$$p \quad - \ point\ in\ given\ cluster$$

$$\left|C_i\right| \quad - \ number\ of\ points\ in\ cluster\ i$$

SSB is a measure of how well separated a cluster is from other clusters. **Total sum of squares (TSS)** is the sum of total SSB and total SSE.

The silhouette coefficient/width is a combination of these two measures, it is defined at a point, cluster and clustering level. It is defined for point ***i*** as follows

$$s(i) = \frac{b(i) - a(i)}{max(b(i), a(i))}$$

$$a(i) - \textit{average distance between point i and all other points in the same cluster}$$

$$b(i) - min_j(\textit{average distance of i to points in cluster j})$$

$$-1 \leq s(i) \leq 1$$

Silhouette width for the cluster is the average of the silhouette width of all points in the cluster. Silhouette width of the clustering is the average of the silhouette width of all points in the dataset. A silhouette width closer to 1 is considered better.

The true SSB, SSE numbers for the above datasets are given below:

- **TwoDimEasy.csv**

Cluster	SSE	SSB	Silhouette Width
Cluster 1	**2.360**	**41.993**	**0.841**
Cluster 2	**16.999**	**35.772**	**0.587**
Total	**19.3582**	**77.7644**	**0.7041**

- **TwoDimHard.csv**

Cluster	SSE	SSB	Silhouette Width
Cluster 1	**0.31284772**	**8.44539896**	**0.73736374**
Cluster 2	**0.90253362**	**2.73443401**	**0.53153513**
Cluster 3	**2.43011872**	**5.23921933**	**0.36306948**
Cluster 4	**1.91071547**	**7.32907332**	**0.44965576**
Total	**5.55621551817**	**23.7481256164**	**0.513143454**

The clustering algorithm was run for both the datasets three times with k value set to the true number of clusters in the dataset.

Here are the findings for TwoDimEasy.csv with **k=2**

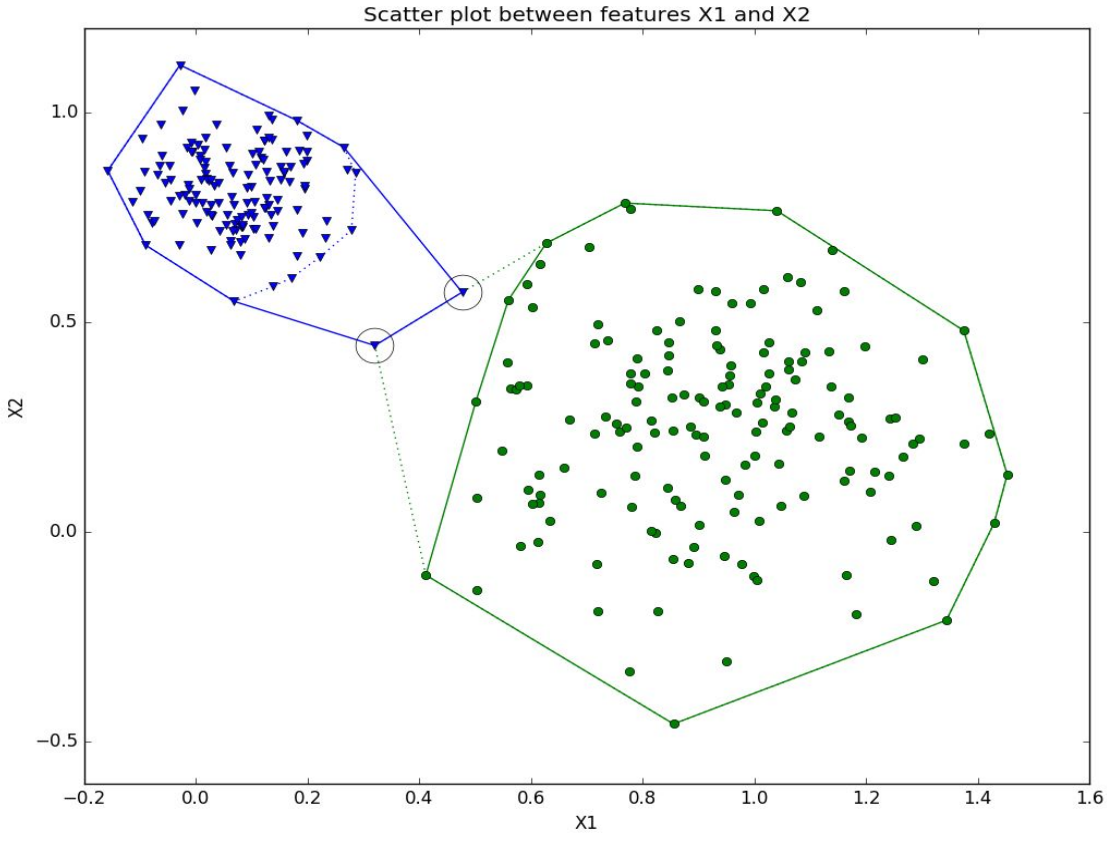
Run 1

Initial centroids: (1.24558023 -0.01997893), (0.61283879 -0.02508299)

Cluster	SSE	SSB	Silhouette Width
Cluster 1	**2.783**	**41.624**	**0.830**
Cluster 2	**16.295**	**36.421**	**0.598**
Total	**19.0778**	**78.0448**	**0.7061**

Confusion matrix:

	1	2
1	**[[1.000**	**0.000]**
2	**[0.012**	**0.988]]**



The **solid lines** indicate the **predicted cluster boundaries** and the **dotted lines** represent the **true cluster boundaries**. The circled points are the misclassified points. We can see from the graph that two

points that originally belonged to cluster 2 are misclassified as cluster 1. This is also evident from the extended boundary cluster 1.

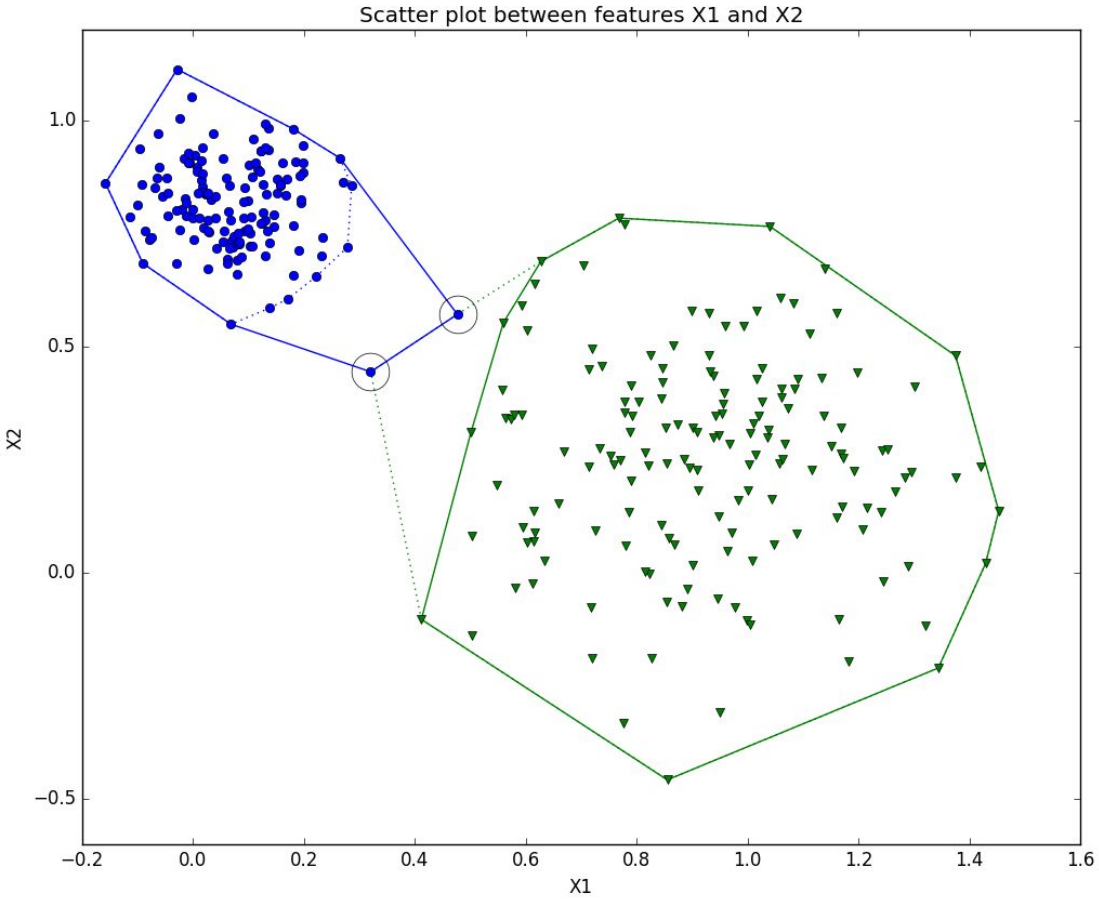
Run 2

Initial Centroids: (0.18107491 0.76588789), (0.93907317 0.29827991)

Cluster	SSE	SSB	Silhouette Width
Cluster 1	**2.783**	**41.624**	**0.830**
Cluster 2	**16.295**	**36.421**	**0.598**
Total	**19.0778**	**78.0448**	**0.7061**

Confusion matrix:

	1	2
1	**[[1.000**	**0.000]**
2	**[0.012**	**0.988]]**



There is not much change in the numbers even though the initial centroids are different.

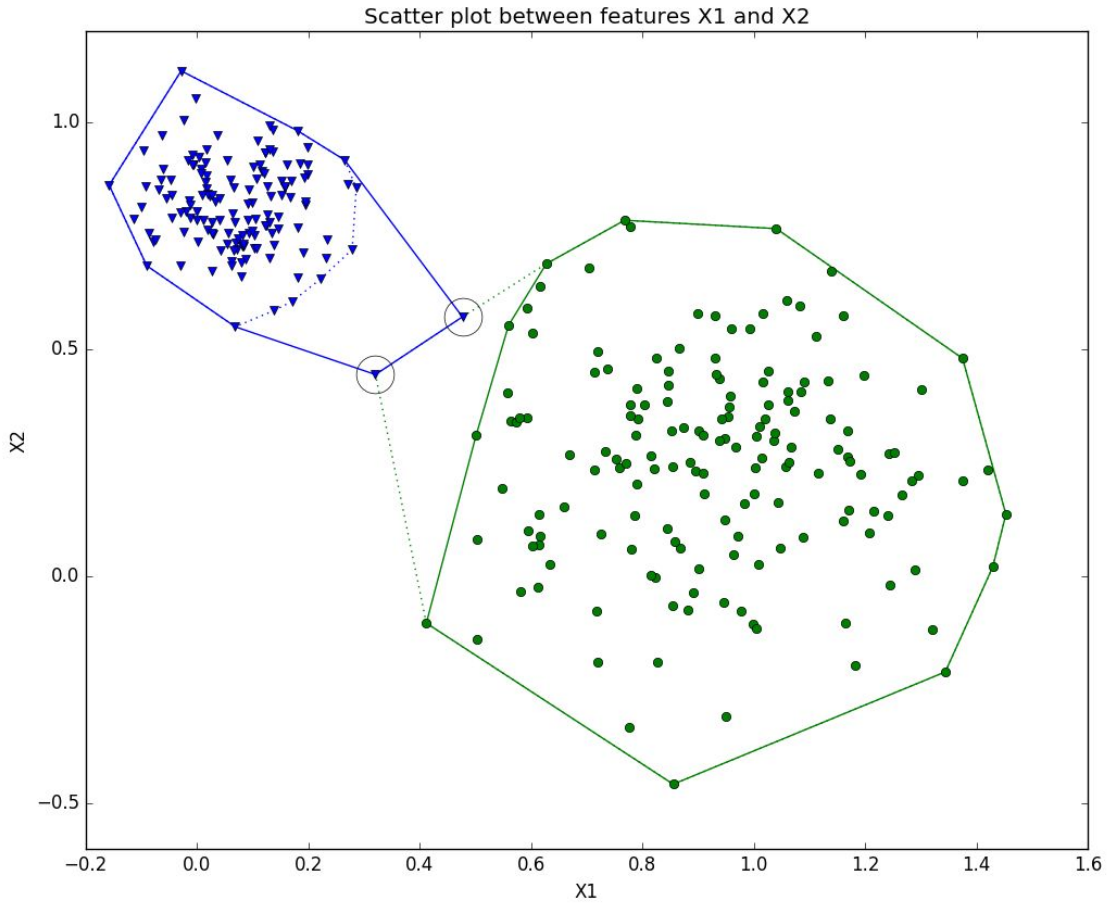
Run 3

Initial Centroids: (1.04310728 0.16257548), (0.01720015 0.85279331)

Cluster	SSE	SSB	Silhouette Width
Cluster 1	2.783	41.624	0.830
Cluster 2	16.295	36.421	0.598
Total	19.0778	78.0448	0.7061

Confusion matrix:

	1	2
1	[[1.000	0.000]
2	[0.012	0.988]]



For the third run as well, the output is exactly the same even though the initial centroids are different.

Here are the findings for TwoDimEasy.csv with **k=4**

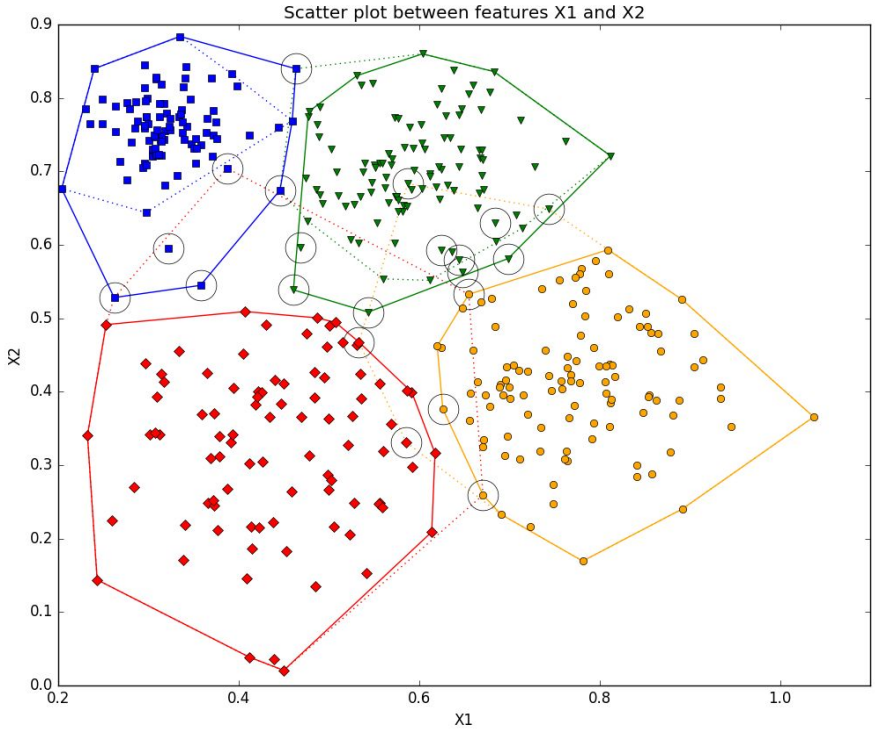
Run 1

Initial Centroids: [[0.81102994, 0.35244979], [0.66871135, 0.52262745], [0.71300129, 0.76912924], [0.86785059, 0.45543119]]

Cluster	SSE	SSB	Silhouette Width
Cluster 1 (yellow)	1.471	7.712	0.519
Cluster 2 (green)	1.076	2.613	0.511
Cluster 3 (red)	1.845	5.503	0.445
Cluster 4 (blue)	0.5	8.584	0.692
Total	4.8921	24.4122	0.5414

Confusion matrix:

	1	2	3	4
1	[[1.000	0.000	0.000	0.000]
2	[0.020	0.980	0.000	0.000]
3	[0.041	0.021	0.907	0.031]
4	[0.000	0.070	0.018	0.912]]



Similar to the scatter plot for the TwoDimEasy case, the solid lines indicate the predicted cluster boundaries and the dotted lines indicate the true cluster boundaries. The misclassified points are circled in the plot. We can see from the confusion matrix that the **Recall/True Positive Rate** for each cluster

number is close 90% or above, with the highest being **100%** for Cluster 1 and the lowest being **90.7%** for Cluster 3.

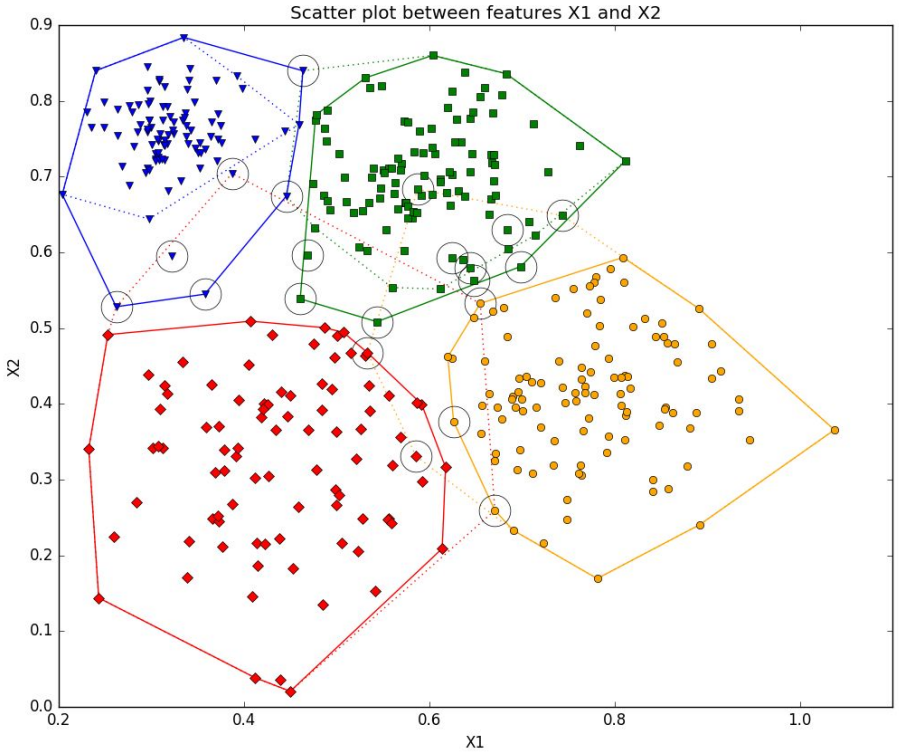
Run 2

Initial Centroids: [[0.67324431 0.39489427], [0.30512627 0.73033242], [0.43465948 0.36563046], [0.46816167 0.59619956]]

Cluster	SSE	SSB	Silhouette Width
Cluster 1 (yellow)	1.471	7.712	0.519
Cluster 2 (blue)	1.076	2.613	0.511
Cluster 3 (red)	1.845	5.503	0.445
Cluster 4 (green)	0.5	8.584	0.692
Total	4.8921	24.4122	0.5414

Confusion matrix:

	1	2	3	4
1	[[1.000	0.000	0.000	0.000]
2	[0.020	0.980	0.000	0.000]
3	[0.041	0.021	0.907	0.031]
4	[0.000	0.070	0.018	0.912]]



The results are same as the first run.

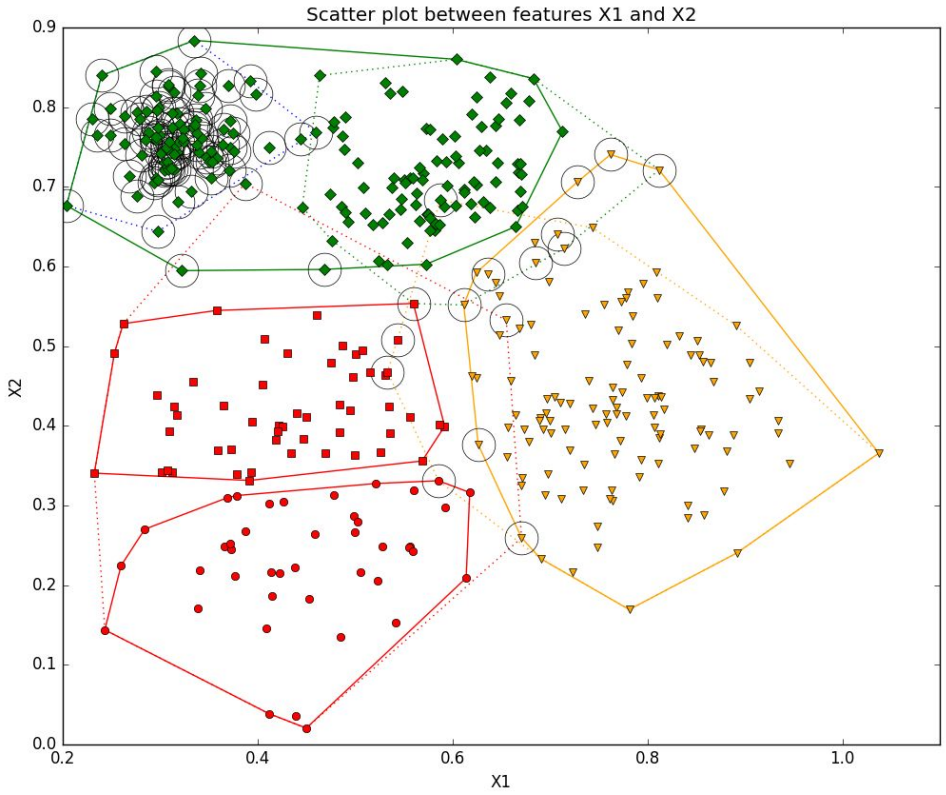
Run 3

Initial Centroids: [[0.40920606 0.14533434] , [0.38793386 0.26726644] , [0.35296152 0.71061786] , [0.36633809 0.24872197]]

Cluster	SSE	SSB	Silhouette Width
Cluster 1 (red)	0.598	4.695	0.329
Cluster 2 (yellow)	2.208	7.455	0.476
Cluster 3 (green)	4.265	7.905	0.452
Cluster 4 (red)	0.641	1.537	0.372
Total	7.712	21.5923	0.4358

Confusion matrix:

	1	2	3	4
1	[[0.000	1.000	0.000	0.000]
2	[0.000	0.910	0.010	0.080]
3	[0.000	0.031	0.938	0.031]
4	[0.000	0.009	0.026	0.965]]



In the third run, we can see that the choice of centroids was very poor. We can see that one true cluster (red in color) in split into two smaller clusters. The two smaller true clusters at the top are combined together into a single bigger predicted cluster, because of which all the points in the true cluster 1 are

marked as misclassified. This is corroborated by the confusion matrix as well. We can see that **recall/True Positive Rate** of cluster 1 points is 0 and all of cluster 1 points are misclassified as cluster 2. As expected, there is an **increase in SSE**, **decrease in SSB** and **decrease in Silhouette width** for this choice of initial centroids.

K = 3 for Datasets 1 and 2

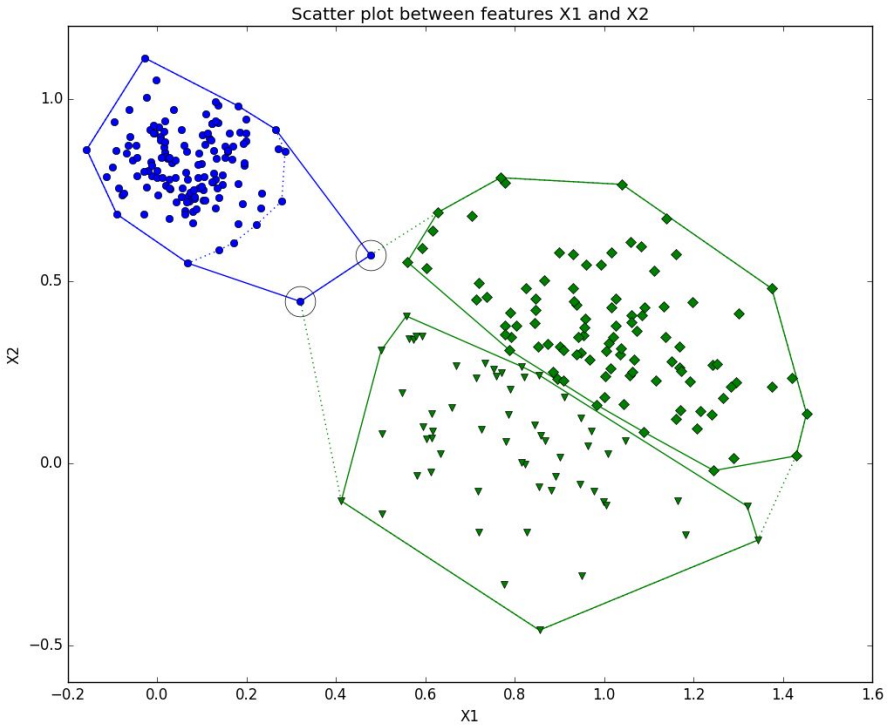
TwoDimEasy

Initial Centroids: [[0.77837129 0.76861648] , [0.61283879 -0.02508299] , [1.01419138 0.2602514]]

Cluster	SSE	SSB	Silhouette Width
Cluster 1 (blue)	2.783	41.624	0.827
Cluster 2 (green)	4.388	16.716	0.3
Cluster 3 (green)	6.537	25.075	0.34
Total	13.7076	83.415	0.5593

Confusion matrix:

	1	2
1	[[1.000	0.000]
2	[0.012	0.988]]



We can see that the overall prediction accuracy has not changed due the increase in the value of k, one of the true clusters is split into two smaller clusters. There is a **slight drop in silhouette width** because the

two predicted clusters (green) are very close to each other. Therefore many points in the these clusters would have very similar intercluster and intracluster average distances. We can see that as **k increases beyond 2** the **silhouette width is bound to decrease**. Therefore, we can conclude that increasing the value of k will not guarantee a better performance for this dataset.

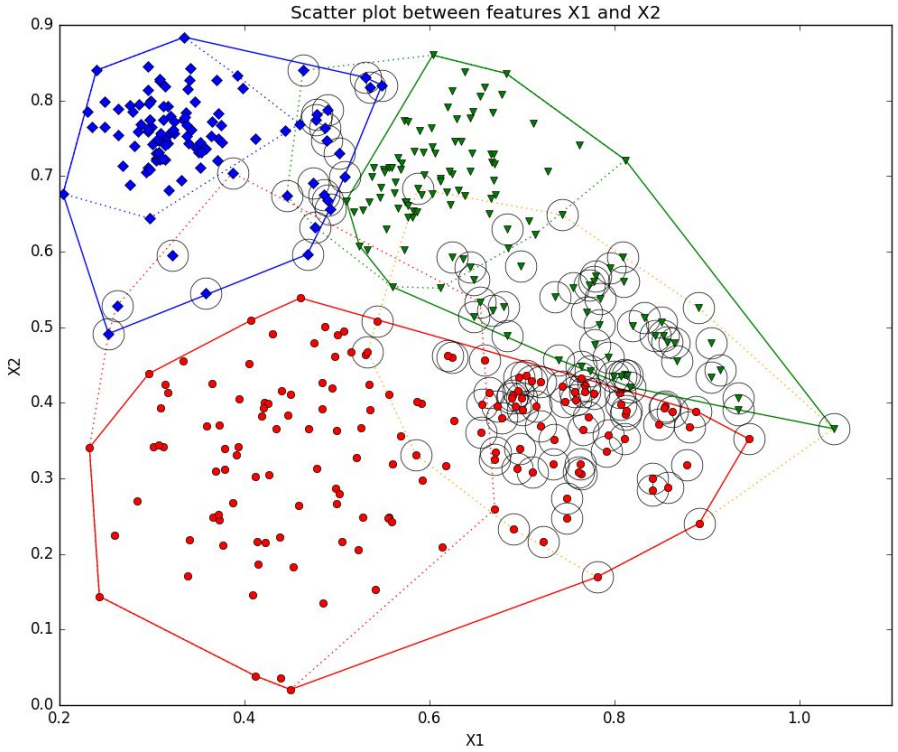
TwoDimHard

Initial Centroids: [[0.65720814 0.39735262], [0.80930698 0.59323796], [0.31121602 0.72250326]]

Cluster	SSE	SSB	Silhouette Width
Cluster 1 (red)	6.132	6.915	0.291
Cluster 2 (green)	3.319	3.136	0.342
Cluster 3 (blue)	1.084	8.718	0.649
Total	10.5358	18.7686	0.408

Confusion matrix:

	1	2	3	4
1	[[1.000	0.000	0.000	0.000]
2	[0.170	0.830	0.000	0.000]
3	[0.062	0.010	0.928	0.000]
4	[0.000	0.412	0.588	0.000]]



The results are as expected, **true cluster 4** is completely misclassified with half of its points being classified as true cluster 3 and half of it being classified as true cluster 2. We can see that the points in

true cluster 1 have a high true positive rate (1.0). This can be explained by the fact that the cluster is very small and choosing any point within the cluster as the initial centroid would result in the cluster being detected properly. We can see that this cluster (**blue**) has the highest silhouette width amongst the three **0.649**.

Decreasing the value of k has drastically impacted the performance of the clustering algorithm. We can conclude that decreasing the value of k below the number of true clusters would result in worse performance. This can be used to guess the number of true clusters that could be present in any given dataset. A consistent drop in the performance would mean that the k value is too less.

Wine dataset

The wine dataset contains 14 features, one of which is the ID feature. It also has a quality and class feature that describe the quality of the wine. The remaining 11 features are used as input to the clustering algorithm. These 11 features are normalized using **min-max normalization** such all the features have the same scale, with the **min as 0 and the max 1**. This ensures that certain features don't dominate the distance measure. Running the k-means clustering algorithm for different values of k on the wine dataset yields the following results:

K	Recall for low class	Recall for High class	Silhouette Width
7	0.672	0.703	0.1965
11	0.675	0.69	0.1489
3	0.69	0.667	0.2149
8	0.731	0.641	0.1722
5	0.761	0.636	0.2002
9	0.726	0.633	0.1497
6	0.777	0.632	0.2109
10	0.761	0.621	0.1431
4	0.778	0.606	0.2077
2	0.698	0.509	0.2445

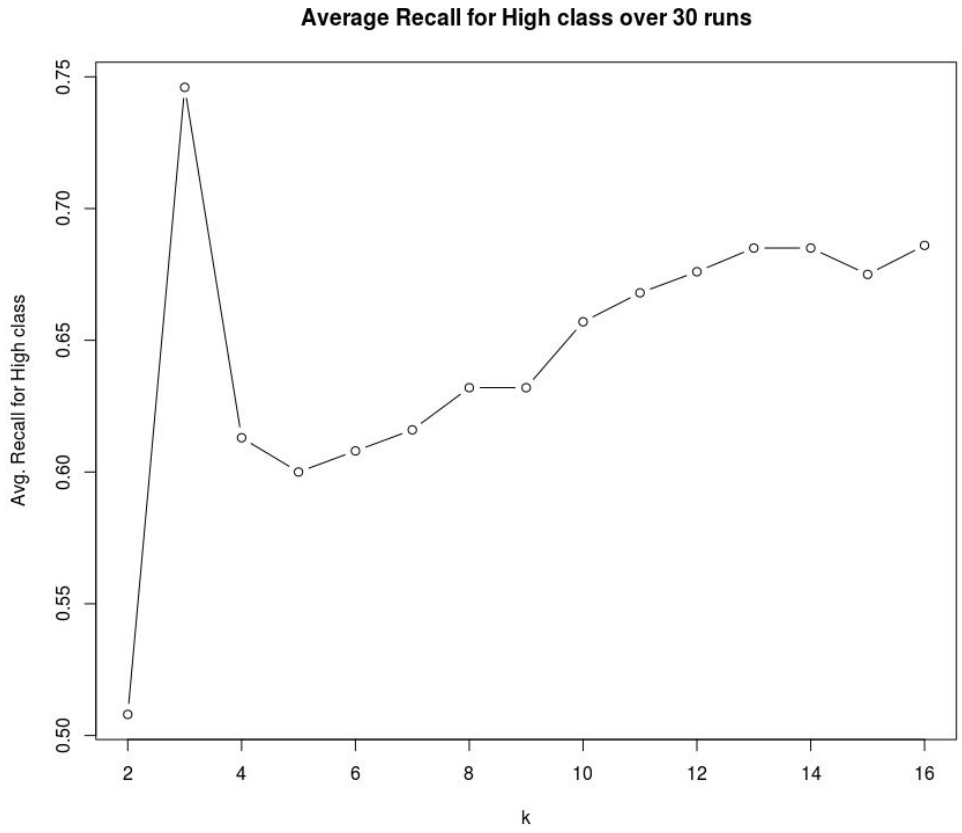
The wine dataset has a class feature associated with each data point which has one of the two values: **High** or **Low**. The clusters identified by the algorithm are classified to either of these two classes by looking at the class label associated with the majority of the points present in the cluster. A confusion matrix was generated using the actual class labels and the predicted class labels. The recall values for the Low class and Recall value for High class in the above table is obtained from this generated confusion matrix.

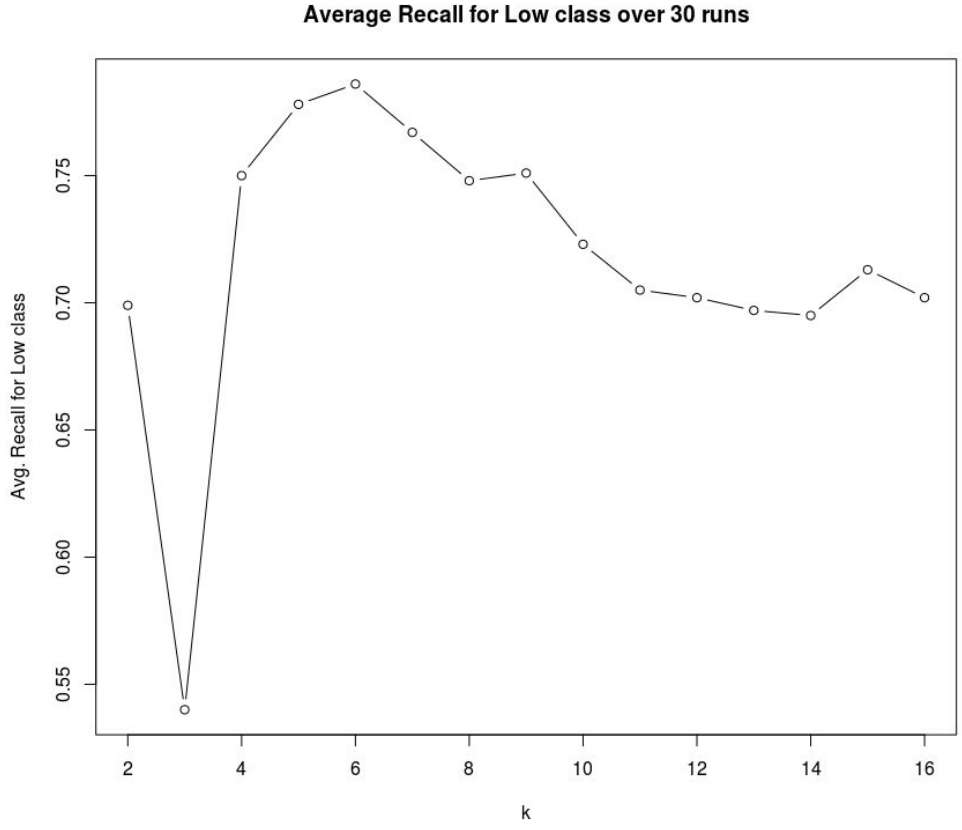
From the above table, it seems like the best value of k, based on the Recall of the High class, is 7. However, this might not be correct as the initial centroids are selected randomly and therefore another run of the algorithm with k=7 could yield poor results. In order to obtain a reliable estimate on the value of k, the algorithm was repeated **30 times** for each value of k, ranging from **2 to 11**. The following table presents the results of these 30 runs. The value for each metric in this table is the average of its value over the 30 runs.

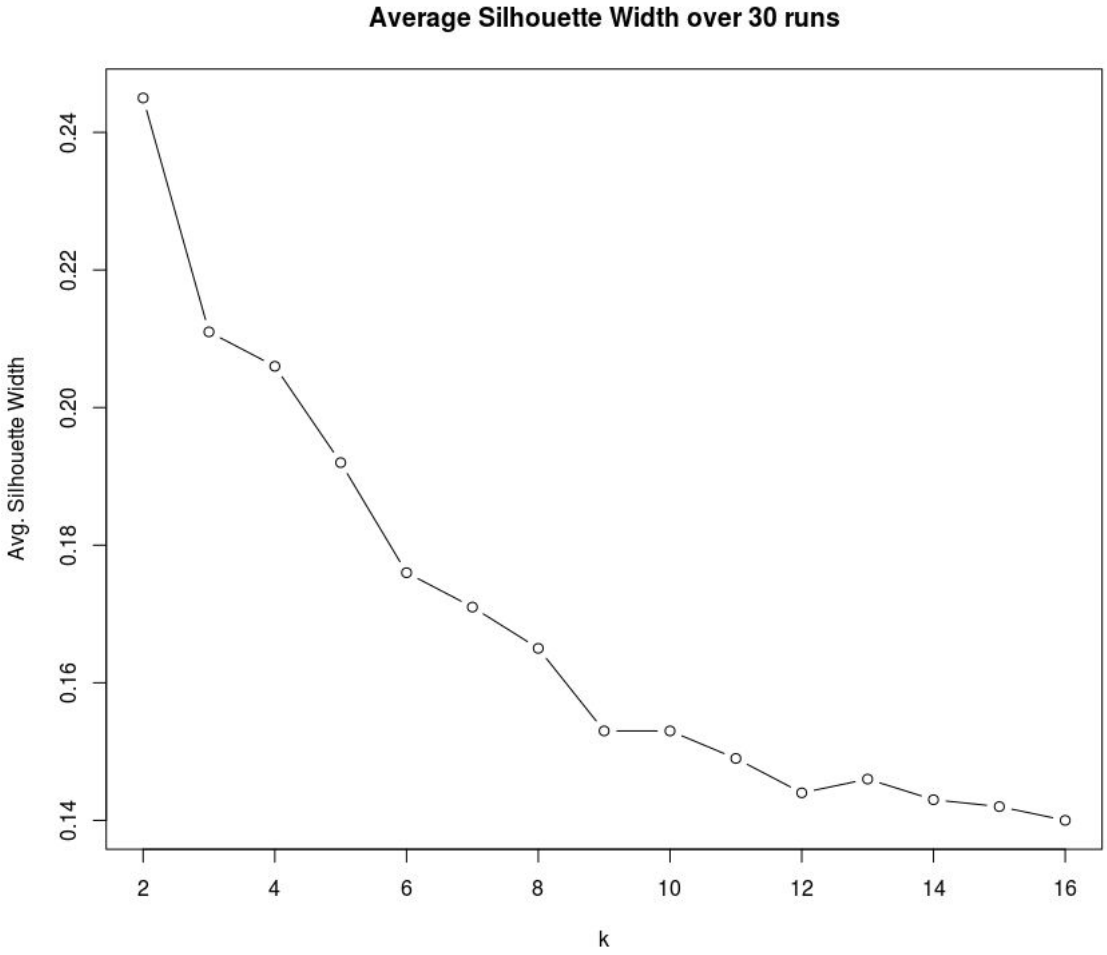
Average metrics obtained after 30 runs of k-means clustering for different values of k

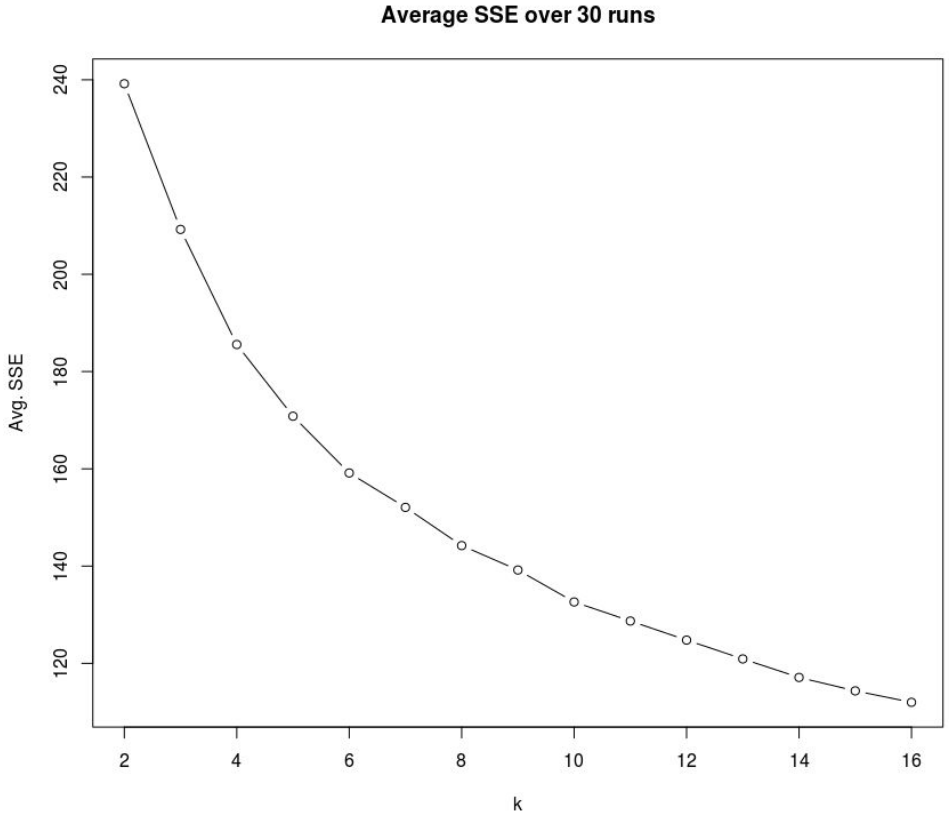
K	Silhouette Width	Recall for low class	Recall for High class	SSE	SSB
3	0.211	0.54	0.746	209.228	108.146
11	0.149	0.705	0.668	128.705	188.669
10	0.153	0.723	0.657	132.623	184.75
8	0.165	0.748	0.632	144.217	173.157
9	0.153	0.751	0.632	139.193	178.18
7	0.171	0.767	0.616	152.072	165.301
4	0.206	0.75	0.613	185.561	131.812
6	0.176	0.786	0.608	159.146	158.228
5	0.192	0.778	0.6	170.819	146.555
2	0.245	0.699	0.508	239.184	78.189

We can see from the above table that **k=3** has the highest avg. Recall for High class but it also has the lowest avg. recall for the Low class. The numbers in the above table are summarized in the following graphs:


Average Recall for High class over 30 runs
Avg. Recall for High class
0.50
0.55
0.60
0.65
0.70
0.75
2
4
6
8
10
12
14
16
k


Average Recall for Low class over 30 runs
Avg. Recall for Low class
0.55
0.60
0.65
0.70
0.75
2
4
6
8
10
12
14
16
k


Average Silhouette Width over 30 runs
Avg. Silhouette Width
0.14
0.16
0.18
0.20
0.22
0.24
2
4
6
8
10
12
14
16
k


Average SSE over 30 runs
Avg. SSE
120
140
160
180
200
220
240
2
4
6
8
10
12
14
16
k

We can also observe that both the **silhouette width and SSE** the clustering has consistently **decreased** as the value of k increases. This could mean that as k increases, new tight clusters are formed very close to each other, resulting in a decrease in overall SSE and inter cluster distance which results in reduction in silhouette width as the intra cluster distance and inter cluster distance become very close to each other.

From the above data, **there is no perfect choice for k**. Choosing **k=3 would give the best recall for high class wines**, but it has a very low recall for low class wines. However, no other choice of k has a recall value more than **0.7** for high class wines.

External validation using the Quality attribute

We can look at the generated clusters for **k=3** and look at how the points are distributed in the cluster. We'll be using the quality attribute of each data point in the formed cluster to understand why the recall rate is very low.

Cluster	Mean quality	Variance
0	5.536	0.623
1	5.994	0.699
2	5.371	0.397

Cluster (k=3)	Number of records for each Quality value					
	3	**4**	**5**	**6**	**7**	**8**
0	7	36	**310**	**301**	56	7
1	2	8	**126**	**221**	**123**	11
2	1	9	**245**	**116**	20	0

From the above numbers, it is evident why the clustering algorithm has a hard time grouping together items of **High (Quality 6-8)** and **Low (Quality 3-5)**. We can see that in all the three clusters formed majority of the records have a quality value of either 5 or 6. This means that almost half of each cluster is

misclassified because a quality value of 6 and above is considered High class and a quality value of 5 and below is labelled Low class. We can look at how these numbers change as the value of k increases.

Cluster (k=7)	Number of records for each Quality value					
	3	**4**	**5**	**6**	**7**	**8**
Cluster 0	0	3	**17**	**106**	87	9
Cluster 1	1	4	**111**	**36**	7	0
Cluster 2	0	7	**27**	**73**	22	4
Cluster 3	7	24	**189**	**131**	21	0
Cluster 4	2	3	**68**	**107**	50	4
Cluster 5	0	11	**162**	**108**	5	1
Cluster 6	0	1	**107**	**77**	7	0

Cluster	Mean Quality	Quality Variance
Cluster 0	6.369	0.548
Cluster 1	5.277	0.376
Cluster 2	5.917	0.692
Cluster 3	5.363	0.586
Cluster 4	5.906	0.692
Cluster 5	5.383	0.369
Cluster 6	5.469	0.332

We can see that even for a higher value of k (k=7), the number of records with quality 5 and quality 6 in majority of the clusters is comparable. Therefore, increasing the value of k doesn't seem to affect the quality of the clusters that are formed.

Part 2

Scipy.kmeans2 clustering method

The scipy kmeans2 clustering method performs k means clustering using the euclidean distance measure. The **kmeans2** library method takes the points, value of k, the initializer as arguments. The initializer value is one of the following:

- **'random'**: generate k centroids from a Gaussian with mean and variance estimated from the data.
- **'points'**: choose k observations (rows) at random from data for the initial centroids.
- **'matrix'**: interpret the k parameter as a k by M (or length k array for one-dimensional data) array of initial centroids.

The **random** initializer was used for the following runs of the clustering method.

TwoDimEasy, k = 2

The results consistently match the output of our k-means clustering implementation.

Predicted cluster level SSE: [16.295 2.783], Predicted overall cluster SSE: 19.0778

Predicted cluster level SSB: [36.421 41.624], Predicted overall cluster SSB: 78.0448

Silhouette width by cluster: [0.598 0.830], Silhouette width: 0.7061

Confusion matrix:

[[1.000 0.000]

 [0.012 0.988]]

TwoDimEasy, k = 3

The clustering method seems to perform better than our implementation. It was able to acheive a 1.0 accuracy for both true clusters consistently

Predicted cluster level SSE: [2.360 6.378 5.128], Predicted overall cluster SSE: 13.8651

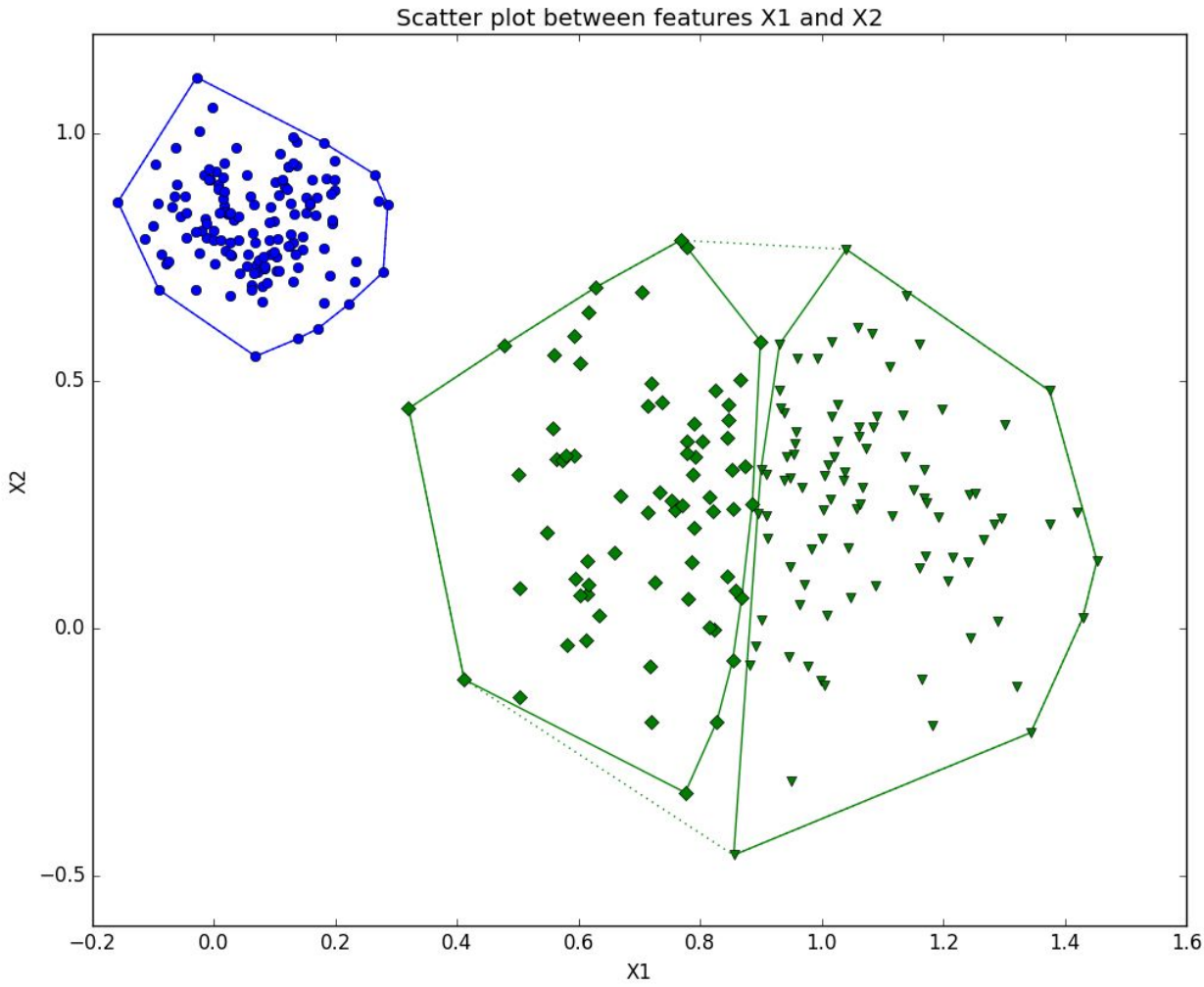
Predicted cluster level SSB: [41.993 34.663 6.601], Predicted overall cluster SSB: 83.2575

Silhouette width by cluster: [0.809 0.318 0.280], Silhouette width: 0.5347

Confusion matrix:

[[1.000 0.000]

 [0.000 1.000]]



TwoDimHard, k=4

The performance of the scipy.kmeans2 clustering method was very similar to our implementation.

Predicted cluster level SSE: [1.076 1.845 1.471 0.500], Predicted overall cluster SSE: 4.8921

Predicted cluster level SSB: [2.613 5.503 7.712 8.584], Predicted overall cluster SSB: 24.4122

Silhouette width by cluster: [0.511 0.445 0.519 0.692], Silhouette width: 0.5414

Confusion matrix:

[[1.000 0.000 0.000 0.000]

[0.020 0.980 0.000 0.000]

[0.041 0.021 0.907 0.031]

[0.000 0.070 0.018 0.912]]

TwoDimHard, k=3

As expected, the output of the clustering method is worse compared to the k=4 case. It is very similar the our implementation of k-means clustering.

Predicted cluster level SSE: [2.208 1.959 4.394], Predicted overall cluster SSE: 8.5614

Predicted cluster level SSB: [7.455 5.470 7.818], Predicted overall cluster SSB: 20.7430

Silhouette width by cluster: [0.503 0.456 0.510], Silhouette width: 0.4955

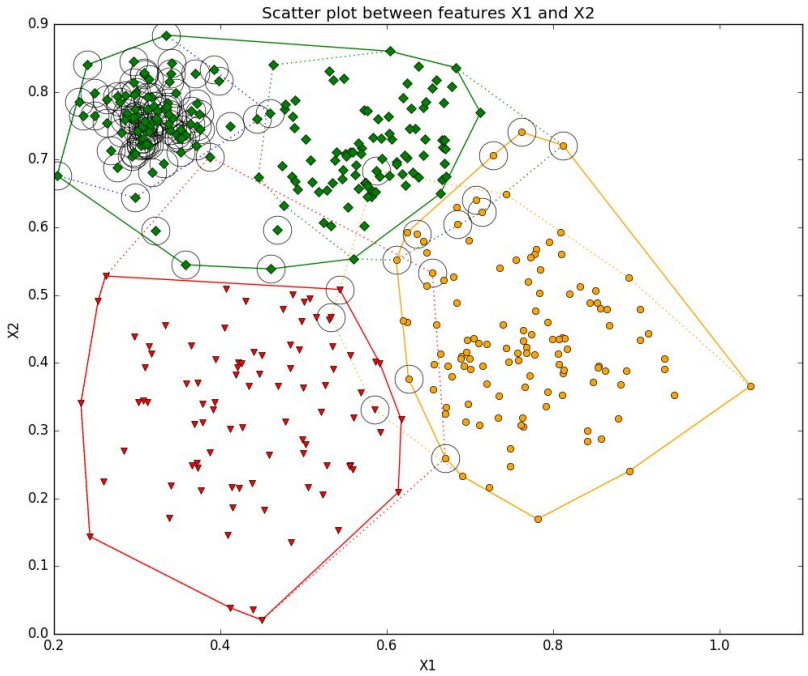
Confusion matrix:

[[0.000 1.000 0.000 0.000]

[0.000 0.920 0.000 0.080]

[0.000 0.052 0.918 0.031]

[0.000 0.009 0.026 0.965]]



BIRCH clustering algorithm

Scikit learn's BIRCH clustering algorithm was also run on the two data sets. This clustering technique has the advantage of not storing the entire input data set in memory. This is ensured by keeping track of the summary statistics of the points, namely, linear sum, squared sum, centroids and the squared norm of the centroids in memory. These statistics are continuously updated as and when a new point is added to the clustering algorithm. These statistics are stored in a tree structure, where a data point belongs to cluster represented by one of the leaf nodes of the tree. All the non-leaf nodes represent a collection of cluster and each node in the tree stores the summary statistics of all the points that belong to any of the clusters that are present under that node in the tree. Therefore, the root node of the tree contains the summary statistics of all the points in the dataset.

Whenever a new point is added to the algorithm, the point starts from the root node of the tree and is pushed down layer by layer until it reaches a leaf node. The summary statistics of the nodes in all these layers are updated as the point is pushed down that layer.

The algorithm defines two settings, namely, **threshold** and **branching factor.** The **branching factor** controls the number of branches allowed at each node in the tree (number of sub clusters allowed at each node). The **threshold** value limits the distance between the entering point and the existing sub clusters. When a new point is added to the tree, the point is merged with all the subclusters present in the root node to determine the subcluster that will have the least radius after merging this new point. The point is then merged to this subcluster. After adding the new point the subcluster, if the radius of the merged subcluster and the next nearest subcluster is greater than the square of the threshold then the subcluster is split into smaller subclusters until the threshold constraint is met. This guarantees that the points within each subcluster are close to each other and also that the points in neighboring sub clusters are not too far apart.

The **n_clusters** argument passed to the BIRCH clustering method is used to generate the final set of clusters from the leaf node sub clusters.

TwoDimEasy, k = 2

With **threshold = 0.1** and **branching_factor=50**, the BIRCH clustering method was able to identify the true clusters consistently without any error.

Predicted cluster level SSE: [16.999 2.360], Predicted overall cluster SSE: 19.3582

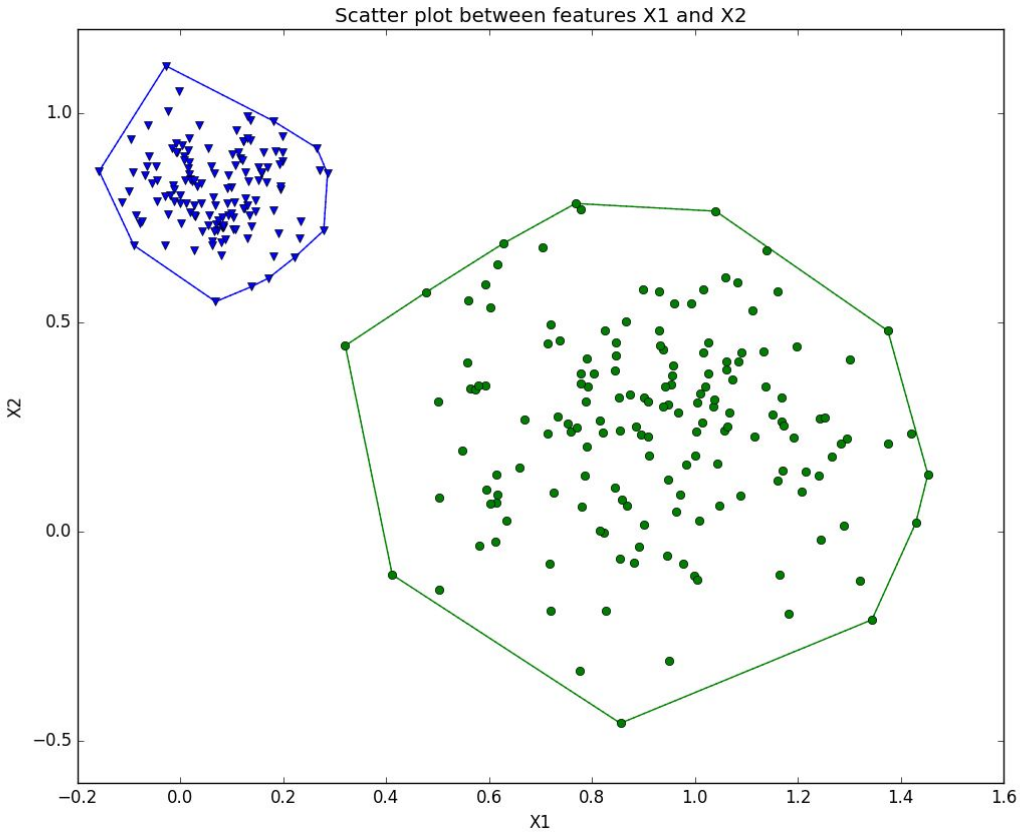
Predicted cluster level SSB: [35.772 41.993], Predicted overall cluster SSB: 77.7644

Silhouette width by cluster: [0.587 0.841], Silhouette width: 0.7041

Confusion matrix:

[[1.000 0.000]

[0.000 1.000]]



TwoDimEasy, k = 3

Predicted cluster level SSE: [7.875 2.360 4.092], Predicted overall cluster SSE: 14.3262

Predicted cluster level SSB: [30.616 41.993 10.187], Predicted overall cluster SSB: 82.7964

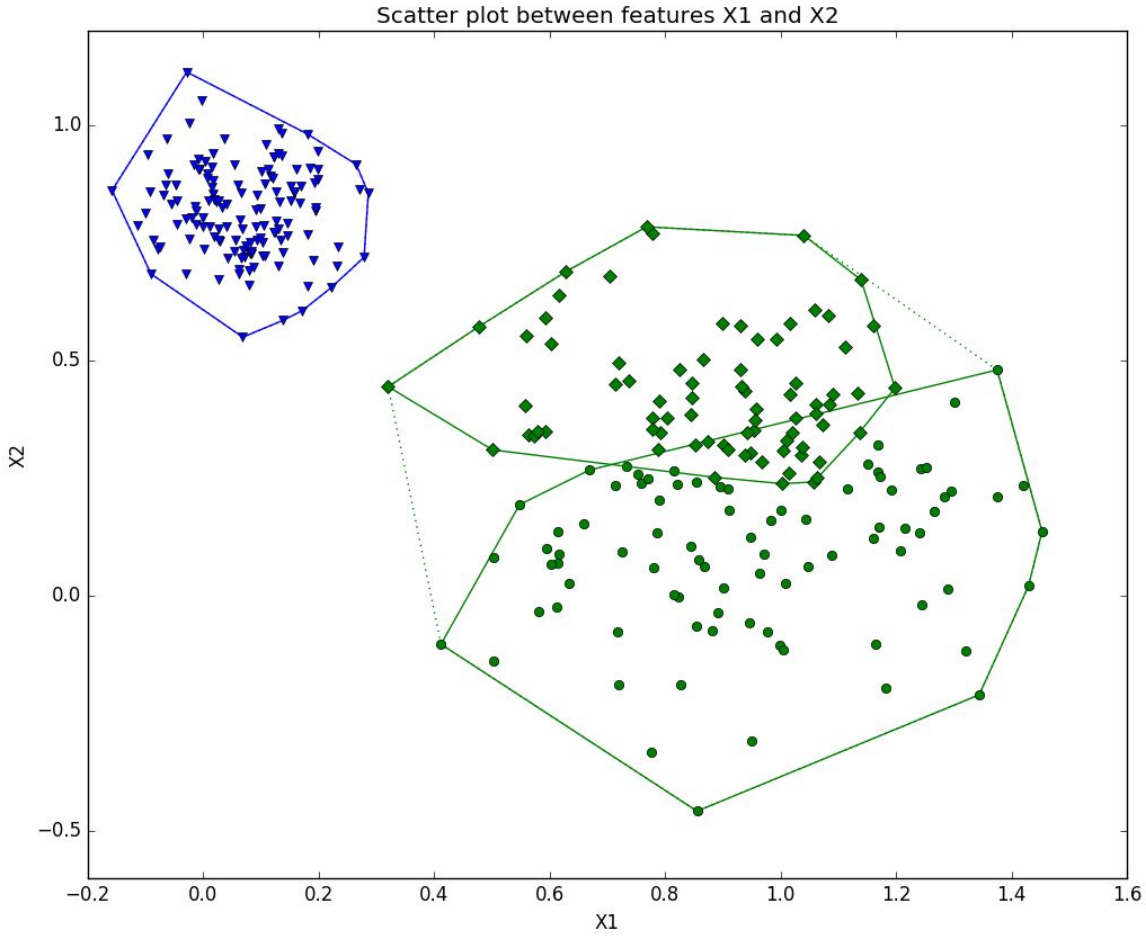
Silhouette width by cluster: [0.164 0.818 0.372], Silhouette width: 0.5181

Confusion matrix:

[[1.000 0.000]

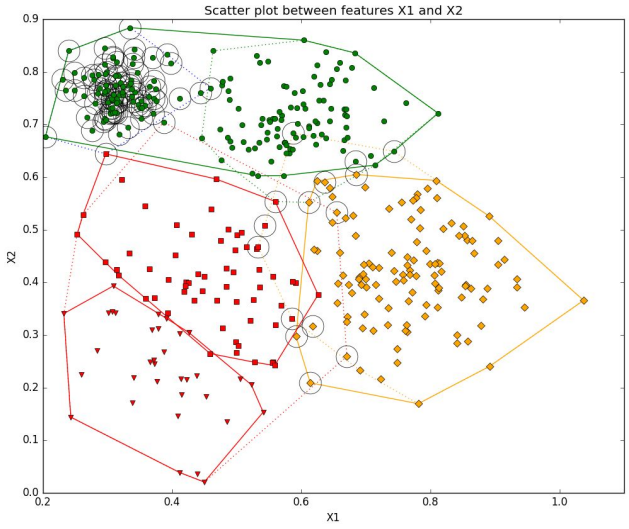
[0.000 1.000]]

Changing the **n_clusters to 3** didn't result in any change in performance. The clustering method was still able to identify the true clusters. However, there is a decrease in silhouette width as the clustering method produces two overlapping clusters. The following scatterplot shows the boundaries of the predicted clusters



TwoDimHard, k = 4

At **threshold=0.1** and **branching_factor=50** the clustering algorithm is not able to find all the clusters, the following scatterplot shows the output of the algorithm for these settings



Changing the **threshold=0.01** results in better clustering as shown in the following numbers and the scatterplot

Predicted cluster level SSE: [2.069 0.839 2.023 0.312], Predicted overall cluster SSE: 5.2429

Predicted cluster level SSB: [5.280 2.826 7.519 8.436], Predicted overall cluster SSB: 24.0615

Silhouette width by cluster: [0.416 0.543 0.440 0.736], Silhouette width: 0.5256

Confusion matrix:

[[0.978 0.022 0.000 0.000]

 [0.000 0.940 0.010 0.050]

 [0.021 0.010 0.928 0.041]

 [0.000 0.009 0.026 0.965]]