CS412: Introduction to Data Mining, Fall 2017, Homework 5

Name: Nestor Alejandro Bermudez Sarmiento (nab6)

Worked individually

Introduction

In this assignment we explore two topics: classifications methods and clustering methods. For classification we will illustrate the use of a Naive Bayes classifier and also K-nearest neighbor using different values of k. Finally we will look at the k-means, DBSCAN and AGNES clustering algorithms.

Question 1

Consider the following equations:

$$Pr(X, C) = Pr(X \mid C) \times Pr(C) \tag{1}$$

$$Pr(C \mid X) = \frac{Pr(X \mid C) \times Pr(C)}{Pr(X)} \tag{2}$$

1a(i): $Pr(\text{Popularity} = \text{'P'}) = \frac{7}{10}$. Done by simply counting the number of examples that match the criteria. Known as the prior probability.

1a(ii): $Pr(\text{Popularity} = \text{'NP'}) = \frac{3}{10}$. Found by taking the value from the previous question from 1 since this is a binary classification problem.

1a(iii): By using equation 1 and class-conditional independence:

$$Pr(\text{Price} = \text{'\$'}, \text{Delivery} = \text{'Yes'} , \text{Cuisine} = \text{'Korean'} \mid \text{Popularity} = \text{'P'})$$

$$= \prod_{i=1}^{3} Pr(x_i \mid \text{Popularity} = \text{'P'})$$

where x_i is each of the features in the tuple considered. So:

$$x_1 = \frac{4}{7}, x_2 = \frac{4}{7}, x_3 = \frac{2}{7}$$

and then

$$Pr(\text{Price} = \text{'\$'}, \text{Delivery} = \text{'Yes'} , \text{Cuisine} = \text{'Korean'} \mid \text{Popularity} = \text{'P'}) = \frac{4}{7} \times \frac{4}{7} \times \frac{2}{7} = \frac{32}{343}$$

1a(iv): Pr(Price = '$', Delivery = 'Yes' , Cuisine = 'Korean' | Popularity = 'NP')

$$= \prod_{i=1}^{3} Pr(x_i \mid \text{Popularity} = \text{'P'})$$

where x_i is each of the features in the tuple considered. So:

$$x_1 = \frac{1}{3}, x_2 = \frac{2}{3}, x_3 = \frac{1}{3}$$

and then

$$Pr(\text{Price} = \text{'\$'}, \text{Delivery} = \text{'Yes'} , \text{Cuisine} = \text{'Korean'} \mid \text{Popularity} = \text{'P'}) = \frac{4}{7} \times \frac{4}{7} \times \frac{2}{7} = \frac{2}{27}$$

1b: When doing Naive Bayes classification one of the common ways to perform the prediction is to using **MAP**[1] (Maximum a posteriori). MAP indicates that we should calculate the probability of each class given the evidence. For this particular case, we have already done that in part 1a(iii) and 1a(iv).

So to answer the question it is just enough to compare the previous results. Since $\frac{32}{343} > \frac{2}{27}$ the answer to the question is **yes**, it will be class as Popular.

1c: the techniques we discussed in class can, theoretically, apply to Naive Bayes. That is: boosting, bagging and ensemble. There is no "one size fit all" solution and finding the right method usually involves trying multiple of them until you get a reasonable accuracy. One of the algorithms we can use is AdaBoost on top of Naive Bayes. This works as follows:

Let d be the number of classes and k the number of iterations or predictors in our ensemble method. These will be the hyper-parameters of our classifier.

1. make all tuples equally likely, e.g. assign the same weight to all of them. The usual value for the weigh is the inverse of the number of classes ($\frac{1}{d}$).
2. create a new data set D_i by sampling with replacement from the original data set. Whether an example is included or not in D_i is decided by its weigh.
3. train a Naive Bayes classifier using D_i.
4. evaluate the Naive Bayes classifier and find its error rate. We can do it by using the test data or by using some of the examples that were not included in D_i.
5. if the error is greater than a threshold (at least 50%), reset the weighs and try again with a new sample (go back to step 2).
6. if the error is good enough then for every tuple in D_i update its corresponding weigh by a factor of the $\frac{e}{acc}$ where e is the error and acc is the accuracy $(1 - e)$.
7. repeat from step 2 if done less than k times.

[1] https://en.wikipedia.org/wiki/Maximum_a_posteriori_estimation

Once the training is done we need to predict labels for new evidence **X**. To do so we initialize the weights with zero and on each iteration we update them by taking the log of the accuracy over the error rate. On each iteration we see what the prediction for the current classifier is and update the weight of that class only! At the end we predict the class with the highest weight.

1d: There are a few metrics that can help measure the performance of a classifier when one of the classes is rare.

i) Sensitivity: it measures the rate of true positives recognition and is defined by the formula: $\frac{TP}{P}$. Where TP is the total number of true positives (correctly classified as positive) and P is the total number of instances that were positive. Note that the formula does not include the negatives so it ignores the fact that the positives are rare because there is no point of comparison between positives and negatives.

ii) Specificity: measures how successful is the classifier in predicting false when over the examples that were indeed false. As an example, in a medical diagnosis, tests usually have high specificity, meaning that they rarely predict a positive in healthy patients, this doesn't say much about whether it was positive or not but it is a first step to perform more tests. Example inspired by this Wikipedia page

Question 2

2a: since $k = 1$ this is rather simple. For every example in the test data we calculate the distance to every exampled seen in the training data. The following table summarizes this and marks the minimum distance in green.

1-NN Distance matrix

Train		Test data							
x1	x2	2.7	2.7	2.5	1	1.5	2.5	1.2	1
1	0.5	2.78		1.58		2.06		0.54	
2	1.2	1.66		0.54		1.39		0.82	
2.5	2	0.73		1.00		1.12		1.64	
3	2	0.76		1.12		1.58		2.06	
1.5	2	1.39		1.41		0.50		1.04	
2.3	3	0.50		2.01		0.94		2.28	
1.2	1.9	1.70		1.58		0.67		0.90	
0.8	1	2.55		1.70		1.66		0.40	

The first 4 rows in the matrix correspond to the training examples that belong to class +1

and the other half the ones that belong to class -1. So, the predicted classes for the test data are: -1, +1, -1 and -1 respectively. And the actual classes are +1, +1, -1, -1 so only the first one was miss-classified and, therefore, the accuracy is $\frac{3}{4} = \mathbf{0.75}$ and the error is **25%**.

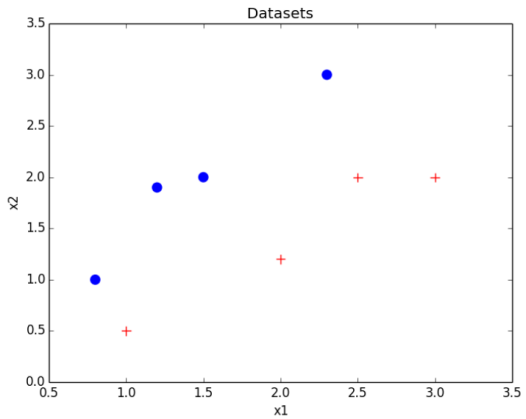
2b: when using $k = 3$ the distance matrix doesn't change but now 3 points "vote" to decide the class to assign to each example. The 3 closest points to each test example are highlighted in green in the following table:

3-NN Distance matrix

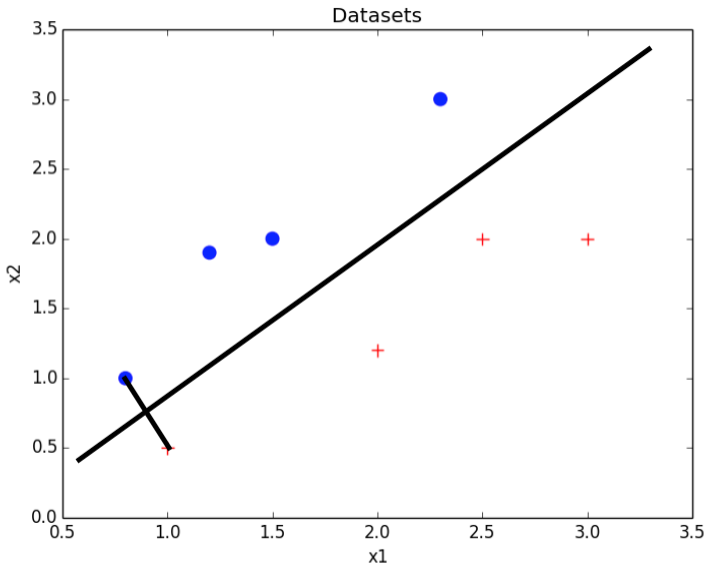
Train		Test data							
x1	x2	2.7	2.7	2.5	1	1.5	2.5	1.2	1
1	0.5	2.78		1.58		2.06		0.54	
2	1.2	1.66		0.54		1.39		0.82	
2.5	2	0.73		1.00		1.12		1.64	
3	2	0.76		1.12		1.58		2.06	
1.5	2	1.39		1.41		0.50		1.04	
2.3	3	0.50		2.01		0.94		2.28	
1.2	1.9	1.70		1.58		0.67		0.90	
0.8	1	2.55		1.70		1.66		0.40	

So for the first test example (2.7, 2.7) two of the three closest points are +1 so it will be classified as +1; for the second example (2.5, 1) all three points are +1 so its class is +1; for (1.5, 2.5) all three are -1 so its class -1 and, finally, (1.2, 1) will be classified as +1.
Only the last prediction is wrong so the accuracy of the model in this case is also **0.75** and the error rate is **25%**.

2c: lets look at the scatter plot shown in the HW description.



What I'll do to find the linear classifier is to take the two points on the bottom-left corner, draw the line that joins them (lets call it L_1), take the midpoint (M_1) and find the perpendicular line (L_2) to L_1 that passes through M_1. The next image illustrates this:



The two points into consideration are (0.8, 1) and (1, 0.5). The first one belongs to class +1. The slope of L_1 is given by

$$m_1 = \frac{x2_+ - x2_-}{x1_+ - x1_-} = \frac{1 - 0.5}{0.8 - 1} = -2.5$$

And the midpoint M_1 is given by

$$(\frac{x1_+ + x1_-}{2}, \frac{x2_+ + x2_-}{2}) = (0.9, 0.75)$$

Finally, to find a perpendicular to a line we need the product of the slopes to be -1. So the slope of L_2 is 0.4. So $x2 = 0.4x1 + b$, when evaluating in the midpoint we have that $0.75 = 0.4 \times 0.9 + b \rightarrow b = 0.39$. As a result the line we are looking for is:

$$f(x) = 0.4x_1 - x_2 + 0.39$$

The following table shows the value resulting of evaluating $f(x)$ on each data example (both training and test).

Linear classifier

x1	x2	f(x)	sign(f(x))
1	0.5	0.29	1
2	1.2	-0.01	-1
2.5	2	-0.61	-1
3	2	-0.41	-1
1.5	2	-1.01	-1
2.3	3	-1.69	-1
1.2	1.9	-1.03	-1
0.8	1	-0.29	-1
2.7	2.7	-1.23	-1
2.5	1	0.39	1
1.5	2.5	-1.51	-1
1.2	1	-0.13	-1

I was expecting to see a perfect accuracy but it wasn't the case. After a while I realized that the problem is that the scale on the axis x1 and x2 in the scatter plot are not equal so, even though L_2 seemed like a good fit it really wasn't. The accuracy for the training data set is $\frac{5}{8} = 62.5\%$ while the accuracy in the test data set is **75%**.

A different way to find $f(x)$ would be to take the same midpoint M_1 we already found and take a second midpoint from the segment that joins (1.2, 1.9) and (2, 1.2). The reason for choosing these two points is that it looks like line drawn before passes through its midpoint. Lets call this point M_2. Its coordinate would be (1.7, 1.95).

Now we just need to find a line that passes through M_1 and M_2. Following the same steps as before we have that $f(x) = 1.5x1 - x2 - 0.6$.

Now lets look at the table evaluating the function on all the examples.

Linear classifier

x1	x2	f(x)	sign(f(x))
1	0.5	0.40	1
2	1.2	1.20	1
2.5	2	1.15	1
3	2	1.90	1
1.5	2	-0.35	-1
2.3	3	-0.15	-1
1.2	1.9	-0.70	-1
0.8	1	-0.40	-1
2.7	2.7	0.75	1
2.5	1	2.15	1
1.5	2.5	-0.85	-1
1.2	1	0.20	1

This time the accuracy on the training data set is **100%** and the accuracy on the test data set is **75%**. Since the question is asking about the error:
Training error is **0%**
Testing error is **25%**

2d: for this particular data set both method perform equally well. In general k Nearest Neighbor is good when you have low dimensional data and many examples but this is not the case here; we do have low dimensional data but also just a few examples. As mentioned earlier in this report, there is no "one size fit all" solution and you usually have to experiment with different methods. Regardless of that, there are some particular scenarios in which one is better than the other. When we have lots of data entries KNN may not be a good fit because the computational complexity could explode.

Question 3

3a: start centroids are (0, 4) and (6, 5). In the first iteration we calculate the distance from the centroids to every point in our data. I'm using Euclidean distance and the results are shown in the following table:

Iteration 0

Examples		Class 1		Class 2		Prediction
x1	x2	0.0	4.0	6.0	5.0	
1	3		1.41		5.39	Class 1
1	2		2.24		5.83	Class 1
2	1		3.61		5.66	Class 1
2	2		2.83		5.00	Class 1
2	3		2.24		4.47	Class 1
3	2		3.61		4.24	Class 1
5	3		5.10		2.24	Class 2
4	3		4.12		2.83	Class 2
4	5		4.12		2.00	Class 2
5	4		5.00		1.41	Class 2
5	5		5.10		1.00	Class 2
6	4		6.00		1.00	Class 2
6	5		6.08		0.00	Class 2

Table 1: Distances from centroid to data points. First iteration.

In this table the green cells indicate the minimum distance between the two cluster centroids. As a result the points (1, 3), (1, 2), (2, 1), (2, 2), (2, 3) and (3, 2) are clustered together and the remaining 7 points in another one.

The next step is to update the centroid by taking the mean of the points included on each cluster. For the first 6 points the mean is **(1.83, 2.17)**, and for the other cluster, the new centroid is **(5, 4.14)**.

Now we need to recalculate the distances using the new centroids:

Iteration 1

Examples		Class 1		Class 2		Prediction
x1	x2	1.83	2.17	5.00	4.14	
1	3		1.18		4.16	Class 1
1	2		0.85		4.54	Class 1
2	1		1.18		4.34	Class 1
2	2		0.24		3.69	Class 1
2	3		0.85		3.21	Class 1
3	2		1.18		2.93	Class 1
5	3		3.27		1.14	Class 2
4	3		2.32		1.52	Class 2
4	5		3.57		1.32	Class 2
5	4		3.66		0.14	Class 2
5	5		4.25		0.86	Class 2
6	4		4.55		1.01	Class 2
6	5		5.04		1.32	Class 2

Table 2: Distances from centroid to data points. Second iteration

After updating the distance we notice that the elements on each cluster remained the same so if we take the means again we would get the same centroids. Because of this, we can stop at this point. So the final result is that the first 6 points belong to cluster 1 and the other 7 to cluster 2.

3b: the first step is to mark all the objects as unvisited and select a random one. Then we calculate the distance from that selected object to all others and count how many there are in the ϵ-neighborhood. See the following table:

Iteration 0

Examples		Distance from		Mark	Cluster
x1	x2	2.00	1.00		
1	3	2.24		Unvisited	
1	2	1.41		Unvisited	
2	1	0.00		Visited	C1
2	2	1.00		Unvisited	
2	3	2.00		Unvisited	
3	2	1.41		Unvisited	
5	3	3.61		Unvisited	
4	3	2.83		Unvisited	
4	5	4.47		Unvisited	
5	4	4.24		Unvisited	
5	5	5.00		Unvisited	
6	4	5.00		Unvisited	
6	5	5.66		Unvisited	

Table 3: Iteration 0 of the DBSCAN clustering algorithm.

The items marked on teal color are the ones that belong to the neighborhood. Since there are more than *MinPts* we will mark the selected point as visited and create our first cluster $C1$. Now we iterate over the points in the neighbor (from top to bottom).

Iteration 1

Examples		Distance from		Mark	Cluster
x1	x2	1.00	2.00		
1	3	1.00		Unvisited	
1	2	0.00		Visited	C1
2	1	1.41		Visited	C1
2	2	1.00		Unvisited	
2	3	1.41		Unvisited	
3	2	2.00		Unvisited	
5	3	4.12		Unvisited	
4	3	3.16		Unvisited	
4	5	4.24		Unvisited	
5	4	4.47		Unvisited	
5	5	5.00		Unvisited	
6	4	5.39		Unvisited	
6	5	5.83		Unvisited	

Table 4: Iteration 1 of the DBSCAN clustering algorithm.

Note that now the distance is calculated from the selected point (1, 2). Since it has 3 objects in its neighborhood we will iterate over those too.

Iteration 2

Examples		Distance from		Mark	Cluster
x1	x2	1.00	3.00		
1	3		0.00	Visited	C1
1	2		1.00	Visited	C1
2	1		2.24	Visited	C1
2	2		1.41	Unvisited	
2	3		1.00	Unvisited	
3	2		2.24	Unvisited	
5	3		4.00	Unvisited	
4	3		3.00	Unvisited	
4	5		3.61	Unvisited	
5	4		4.12	Unvisited	
5	5		4.47	Unvisited	
6	4		5.10	Unvisited	
6	5		5.39	Unvisited	

Table 5: Iteration 2 of the DBSCAN clustering algorithm.

The objects in (1, 3) neighborhood were already included so we keep going.

Iteration 3

Examples		Distance from		Mark	Cluster
x1	x2	2.00	2.00		
1	3		1.41	Visited	C1
1	2		1.00	Visited	C1
2	1		1.00	Visited	C1
2	2		0.00	Visited	C1
2	3		1.00	Unvisited	
3	2		1.00	Unvisited	
5	3		3.16	Unvisited	
4	3		2.24	Unvisited	
4	5		3.61	Unvisited	
5	4		3.61	Unvisited	
5	5		4.24	Unvisited	
6	4		4.47	Unvisited	
6	5		5.00	Unvisited	

Table 6: Iteration 3 of the DBSCAN clustering algorithm.

Iteration 4

Examples		Distance from		Mark	Cluster
x1	x2	2.00	3.00		
1	3		1.00	Visited	C1
1	2		1.41	Visited	C1
2	1		2.00	Visited	C1
2	2		1.00	Visited	C1
2	3		0.00	Visited	C1
3	2		1.41	Unvisited	
5	3		3.00	Unvisited	
4	3		2.00	Unvisited	
4	5		2.83	Unvisited	
5	4		3.16	Unvisited	
5	5		3.61	Unvisited	
6	4		4.12	Unvisited	
6	5		4.47	Unvisited	

Table 7: Iteration 4 of the DBSCAN clustering algorithm.

Iteration 5

Examples		Distance from		Mark	Cluster
x1	x2	3.00	2.00		
1	3		2.24	Visited	C1
1	2		2.00	Visited	C1
2	1		1.41	Visited	C1
2	2		1.00	Visited	C1
2	3		1.41	Visited	C1
3	2		0.00	Visited	C1
5	3		2.24	Unvisited	
4	3		1.41	Unvisited	
4	5		3.16	Unvisited	
5	4		2.83	Unvisited	
5	5		3.61	Unvisited	
6	4		3.61	Unvisited	
6	5		4.24	Unvisited	

Table 8: Iteration 5 of the DBSCAN clustering algorithm.

Since (4, 3) hasn't been included so far, we will include it.

Iteration 6

Examples		Distance from		Mark	Cluster
x1	x2	4.00	3.00		
1	3		3.00	Visited	C1
1	2		3.16	Visited	C1
2	1		2.83	Visited	C1
2	2		2.24	Visited	C1
2	3		2.00	Visited	C1
3	2		1.41	Visited	C1
5	3		1.00	Unvisited	
4	3		0.00	Visited	C1
4	5		2.00	Unvisited	
5	4		1.41	Unvisited	
5	5		2.24	Unvisited	
6	4		2.24	Unvisited	
6	5		2.83	Unvisited	

Table 9: Iteration 6 of the DBSCAN clustering algorithm.

Iteration 7

Examples		Distance from		Mark	Cluster
x1	x2	5.00	3.00		
1	3		4.00	Visited	C1
1	2		4.12	Visited	C1
2	1		3.61	Visited	C1
2	2		3.16	Visited	C1
2	3		3.00	Visited	C1
3	2		2.24	Visited	C1
5	3		0.00	Visited	C1
4	3		1.00	Visited	C1
4	5		2.24	Unvisited	
5	4		1.00	Unvisited	
5	5		2.00	Unvisited	
6	4		1.41	Unvisited	
6	5		2.24	Unvisited	

Table 10: Iteration 7 of the DBSCAN clustering algorithm.

Iteration 8

Examples		Distance from		Mark	Cluster
x1	x2	5.00	4.00		
1	3		4.12	Visited	C1
1	2		4.47	Visited	C1
2	1		4.24	Visited	C1
2	2		3.61	Visited	C1
2	3		3.16	Visited	C1
3	2		2.83	Visited	C1
5	3		1.00	Visited	C1
4	3		1.41	Visited	C1
4	5		1.41	Unvisited	
5	4		0.00	Visited	C1
5	5		1.00	Unvisited	
6	4		1.00	Unvisited	
6	5		1.41	Unvisited	

Table 11: Iteration 8 of the DBSCAN clustering algorithm.

At this point we can see that all the unvisited elements will eventually be assigned to cluster $C1$ so we are done.

Final state

Examples		Distance from		Mark	Cluster
x1	x2	6.00	5.00		
1	3		5.39	Visited	C1
1	2		5.83	Visited	C1
2	1		5.66	Visited	C1
2	2		5.00	Visited	C1
2	3		4.47	Visited	C1
3	2		4.24	Visited	C1
5	3		2.24	Visited	C1
4	3		2.83	Visited	C1
4	5		2.00	Visited	C1
5	4		1.41	Visited	C1
5	5		1.00	Visited	C1
6	4		1.00	Visited	C1
6	5		0.00	Visited	C1

Table 12: Final state of the DBSCAN clustering algorithm.

This result matches the result obtained by using sklearn with the same hyper-parameters. The code can be found together with this report.

3c: consider the initial state

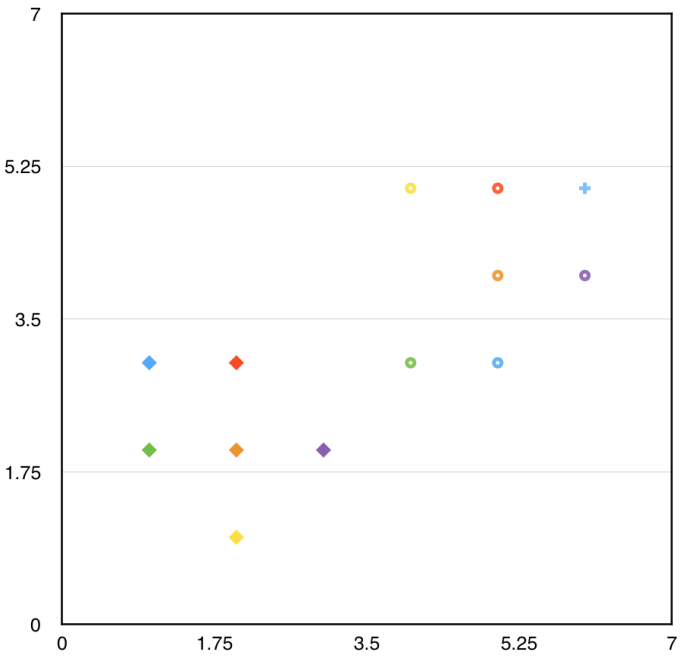


Figure 1: Initial state for the AGNES clustering algorithm.
A cluster is considered the same if it has the same shape and color. So in this plot, all 13 points belong to a different cluster.

The dissimilarity matrix is as follows:

Point		x1	x2	x1	x2	x1	x2	x1	x2	x1	x2	x1	x2	x1	x2	x1	x2	x1	x2	x1	x2	x1	x2	x1	x2
x1	x2	1	3	1	2	2	1	2	2	2	3	3	2	5	3	4	3	4	5	5	4	5	5	6	4	6	5
1	3	0.000	**1.000**	2.236	1.414	**1.000**	2.236	4.000	3.000	3.606	4.123	4.472	5.099	5.385
1	2	**1.000**	0.000	1.414	**1.000**	1.414	2.000	4.123	3.162	4.243	4.472	5.000	5.385	5.831
2	1	2.236	1.414	0.000	**1.000**	2.000	1.414	3.606	2.828	4.472	4.243	5.000	5.000	5.657
2	2	1.414	**1.000**	**1.000**	0.000	**1.000**	**1.000**	3.162	2.236	3.606	3.606	4.243	4.472	5.000
2	3	**1.000**	1.414	2.000	**1.000**	0.000	1.414	3.000	2.000	2.828	3.162	3.606	4.123	4.472
3	2	2.236	2.000	1.414	**1.000**	1.414	0.000	2.236	1.414	3.162	2.828	3.606	3.606	4.243
5	3	4.000	4.123	3.606	3.162	3.000	2.236	0.000	**1.000**	2.236	**1.000**	2.000	1.414	2.236
4	3	3.000	3.162	2.828	2.236	2.000	1.414	**1.000**	0.000	2.000	1.414	2.236	2.236	2.828
4	5	3.606	4.243	4.472	3.606	2.828	3.162	2.236	2.000	0.000	1.414	**1.000**	2.236	2.000
5	4	4.123	4.472	4.243	3.606	3.162	2.828	**1.000**	1.414	1.414	0.000	**1.000**	**1.000**	1.414
5	5	4.472	5.000	5.000	4.243	3.606	3.606	2.000	2.236	**1.000**	**1.000**	0.000	1.414	**1.000**
6	4	5.099	5.385	5.000	4.472	4.123	3.606	1.414	2.236	2.236	**1.000**	1.414	0.000	**1.000**
6	5	5.385	5.831	5.657	5.000	4.472	4.243	2.236	2.828	2.000	1.414	**1.000**	**1.000**	0.000

Table 13: Dissimilarity matrix for all points.

The bold numbers are the ones with minimum dissimilarity. After merging we have:

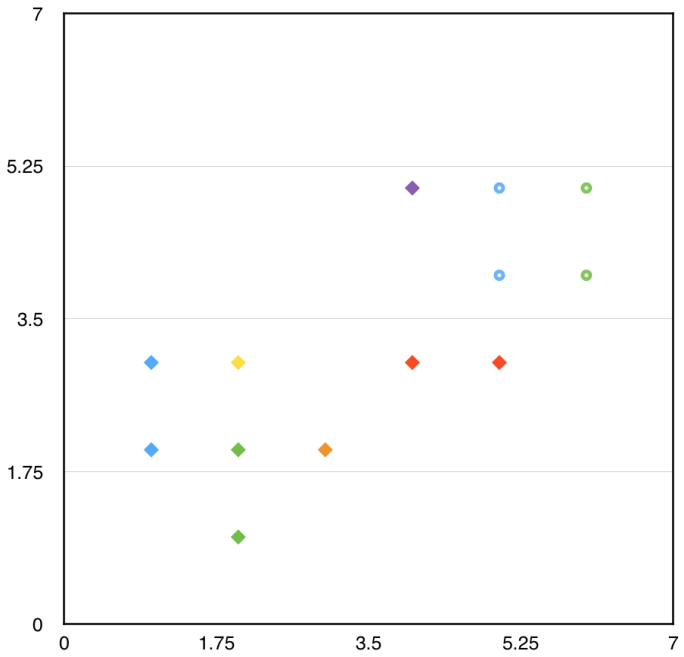


Figure 2: First merge of clusters.

The next iteration results in:

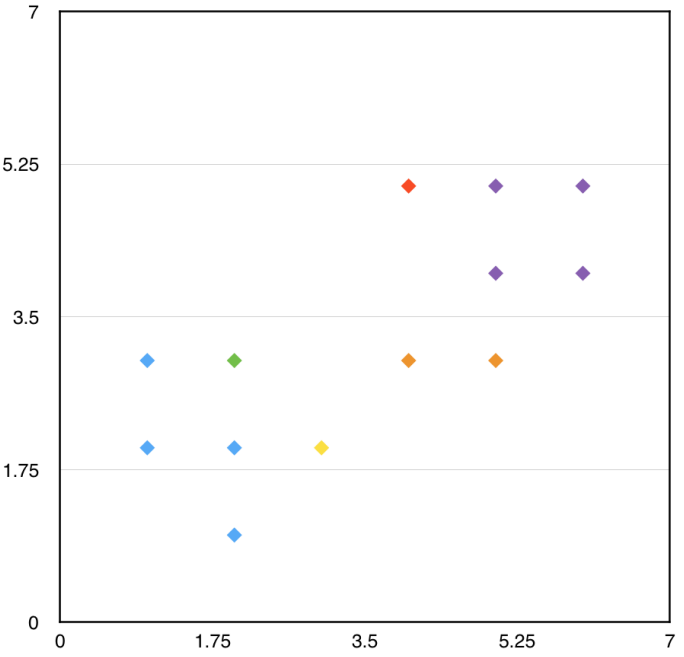


Figure 3: Second merge of clusters.

And for the next one we merge two clusters twice:

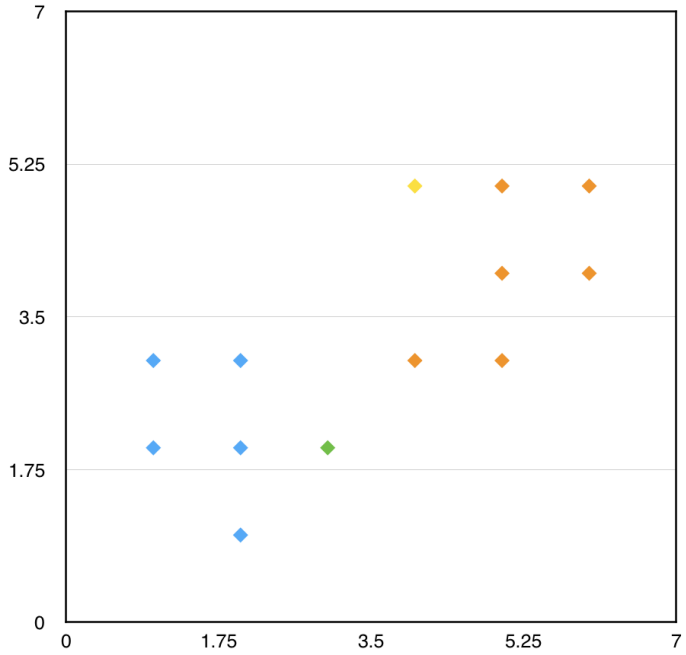


Figure 4: Third merge of clusters.

On the fourth iteration we finally have just two clusters that match the ones found in part 3a.

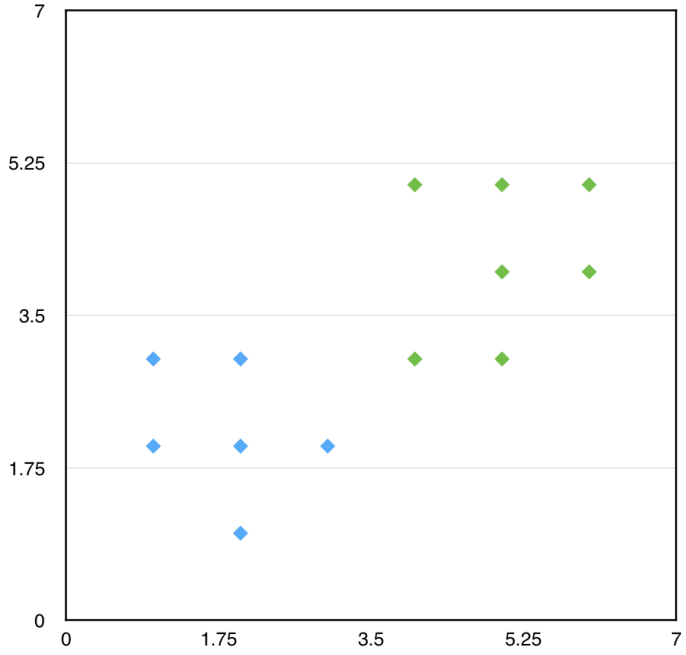


Figure 5: Fourth merge of clusters.

We could perform one more merge and end up with a single cluster. I won't do it because it

is obvious what the result is and the process is already evident.

Finally, below there is the dendrogram to illustrate the merging of the different clusters. At the lowest level, cluster i is the i-th element in our data.

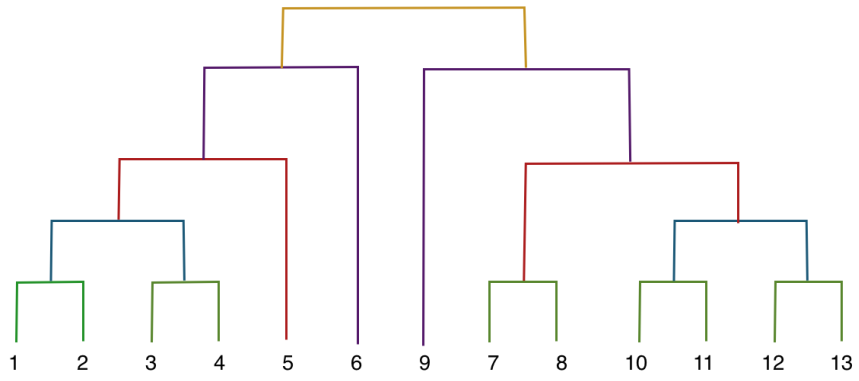


Figure 6: Dendrogram for our AGNES algorithm.