COMS21202: Symbols, Patterns and Signals Report

James Foster (jf16688), Yi-Ching Chen (yc16011)

March 19, 2018

1 Introduction

The processes of unsupervised and supervised learning are quite significant in machine learning. For example, if we know that all the fruit in a particular basket falls into one of two classes, apples or oranges, we want to be able to identify the class of any fruit we pick. We take a sample of fruits from the baskets which serves as an unlabelled data set, which we then group into different classes by selecting the relevant features, such as colour and brightness, and run a clustering algorithm on them. Now that there are labelled sets of data, we can carry out a supervised learning algorithm and build up a classifier. The aim of this coursework is for us to understand the concept of clustering using K-means and classification using nearest centroid and maximum-likelihood. This report will demonstrate in detail how we perform clustering and classification.

2 Feature selection

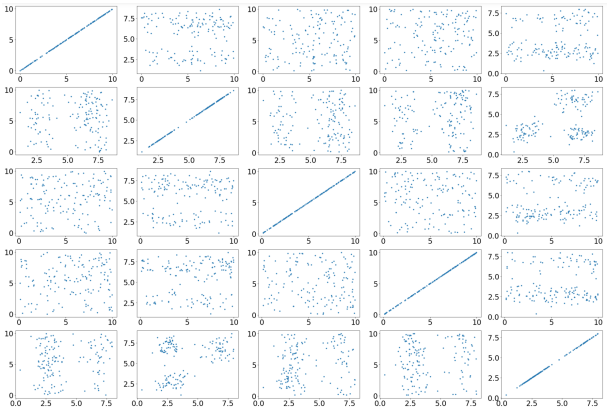

Figure 1: Five features plotted against each other

From the given training data set of 150 unlabelled points, we plot a 5x5 table of small graphs which each have two features plotted against each other. This tells us that there are five different features in total. Where the features are plotted against themselves the line on the graph follows y = x, because x and y are the same.

Our first task is to select the two best features to classify the data. We're looking for two features that clearly separate out the data into multiple clusters, with the higher number of clusters the better. This is because if we have more clusters we can put the data points into more precise classes. On most of the graphs the points are scattered all around the plane fairly randomly, but the points on the plot for features 2 and 5 are gathered into 3 well spaced out clusters, so we choose them to be the features that separate the classes.

Then we stored the second and fifth columns of the data set into a 150 x 2 matrix $\mathbf{X}$.

3 Identifying the classes

After we have the features selected, we need to categorize the data points into three separate classes. In order to do this, we first need to apply the K-means algorithm to the matrix $\mathbf{X}$.

K-means clustering works by first initialising K centroids (either randomly or manually), assigning each data point to its nearest centroid, recomputing the centroids to be the mean of the data points assigned to them, and then repeating this assigning and recomputation until the centroids don't change from one iteration to the next, or the maximum number of iterations is reached (usually 300).

In this case we have identified 3 clusters, so we'll set K to 3 and run the K-means algorithm. This returns three values: a list of

3 centroids, a list of 150 labels, and the inertia of this configuration. The labels identify which data point maps to which cluster and the inertia is the summation of the within-cluster scatter for each cluster. The within-cluster scatter is calculated by finding the total squared distance between all the data points in a cluster and the centroid of that cluster. K-means seeks to minimise the inertia, such that a local optimum is found where no improvements can be made by continuing to run the algorithm and the final stable configuration produces well separated clusters.

Once we've run K-means we can plot a scatter graph of our data, colouring each data point according to its corresponding label and then plotting the 3 centroids in red to easily identify them.

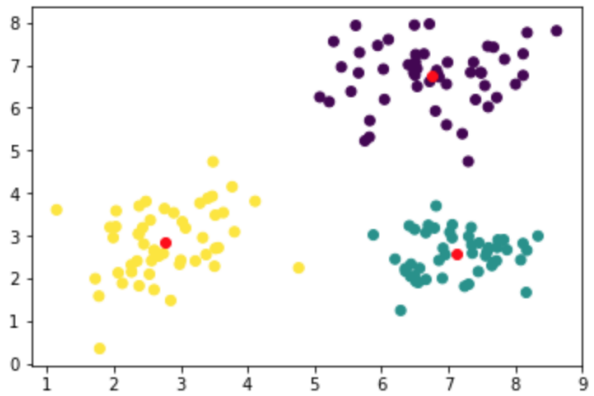

Figure 2: 3 clusters identified

4 Nearest-centroid classification

To find a nearest-centroid classifier for our clusters, we plot a Voronoi diagram, which partitions the plane according to the centroids, such that all the points in one centroid's region will be closer to that centroid than any other one. We then add our test data to see which class they fall into. We colour-code them according to the nearest centroid and represented them with stars.

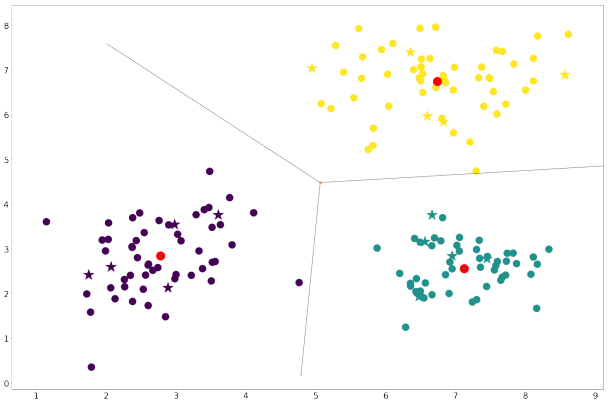

Figure 3: Voronoi diagram

From the diagram we can see that the algorithm separates the clusters quite well and that the training data points would all be assigned to the same cluster as they were for K-means.

If we want to find a non-optimal clustering, we have two main strategies:

1. We set the initial centroids manually. To find the 'bad' centroids, we run K-means many times with randomly generated vectors as the initial centroids, and pick one where the inertia is greater than 500. Our optimal solution had an inertia of 152, so a solution with a significantly higher inertia like 500 will be far below optimal.

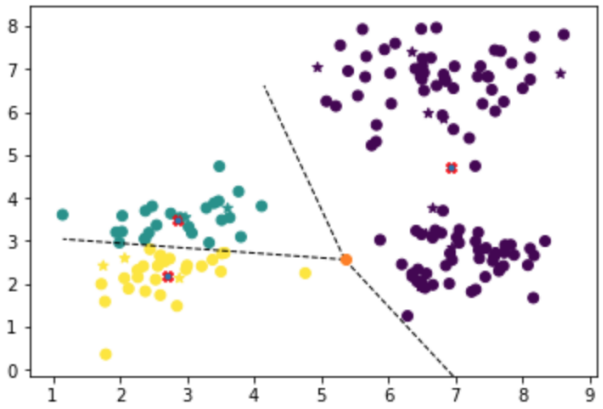

Figure 4: Non-optimal clustering with bad initial centroids

2. We again use random initial centroids, but instead we perform a massively reduced number of iterations and set the number of different initial centroid seeds to 1, because we don't want the algorithm trying many different starting configurations and picking the best one, we want to find the worst one possible. To illustrate the extreme end we'll only perform 1 iteration. By its random nature this option is inconsistent. If the initial centroids are close to the optimal ones the iner-

tia will be low and we'll get decision boundaries that well separate the clusters, but if they're far away then 1 iteration won't be enough to correct for this, so the inertia is high and the decision boundaries are poor.

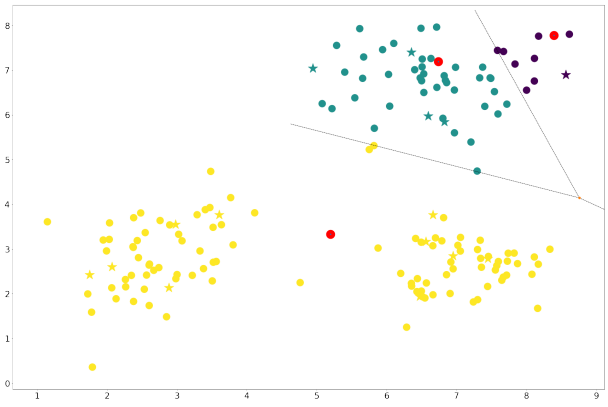

Figure 5: Non-optimal clustering with one iteration

5 Maximum-likelihood classification

To create a Maximum-Likelihood (ML) classifier, we first need to model each cluster with a 1-dimensional random variables **Z**. In this case, we assume it follows a bivariate normal distribution (aka. a 2D Gaussian distribution) centred around its centroid, with the covariance matrix for the cluster determining the specific proportions of the distribution.

$$Z = \frac{1}{det(2\pi\Sigma)^{\frac{1}{2}}} e^{\frac{-x\Sigma^{-1}x^T}{2}}$$

The probability density function (PDF) for **Z** can be used to find the relative likelihood of **Z** at any given point. If we feed the PDF a set of points that covers the entire space the data occupies, in this case (0, 0) to (9, 9), we can build up a 3D bell curve of the probability density. To represent it in 2D space we use a contour plot, which uses lines to show where **Z** is at a particular value. They usually have multiple contour lines to show the slope of the 3D graph, but we're only looking to find the contour level where the probability mass inside it adds to a 95%.

To do this, we need to find a value of t where:

$$P(Z \geq t) = 0.95$$

Given:

$$Z' = x\Sigma^{-1}x^T$$

and

$$P(Z \geq t) = P(Z' \leq t') = 0.95$$

We can say that

$$P(Z' \geq t') = 0.05$$

Because Z' follows the Chi Squared distribution with degree of freedom 2, we can look up the value of t' for when the P value is 0.05, which comes out to 5.99. If we then substitute 5.99 for Z' in our original equation for Z, we can find the value of t. Therefore:

$$t = \frac{1}{det(2\pi\Sigma)^{\frac{1}{2}}} e^{-\frac{5.99}{2}}$$

Finally we set our contour plot to have t as its only contour level, giving us our desired ellipse containing 95% of the probability mass for the cluster. The size and shape of the ellipse will depend on the covariance matrix of the cluster.

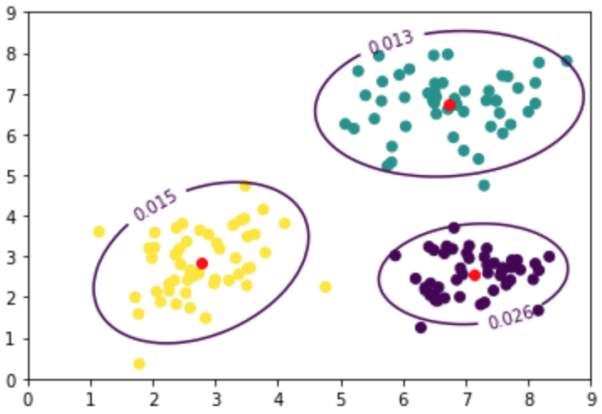

Figure 6: Contour plot with 95% level

We find the pairwise likelihood between 2 classes by first getting the ratio of their PDFs at each point in the plane. Since this involves dividing by numbers very close to 0 we get some very large ratios, which makes it hard to make a good contour plot for, so we cap the ratio at 2 since the magnitude of the ratio is unimportant for our purposes. After that, we make a contour plot of the ratios with a single contour level of 1. This draws a line where the 2 classes are of equal likelihood, giving us a decision boundary between them. We then repeat this for every combination of classes.

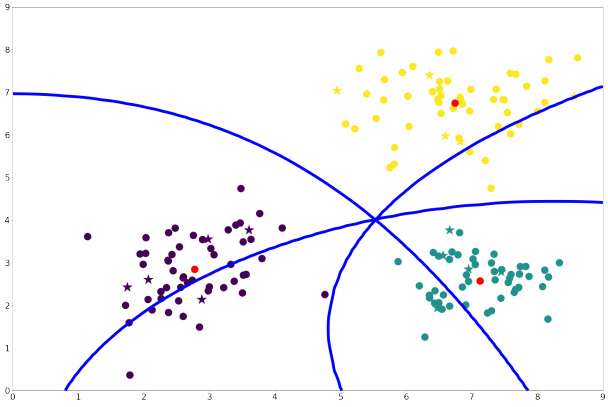

Figure 7: Maximum-likelihood classifier

We can see that there are 3 decision boundaries for the 3 different combinations of clusters and each of them is curved, some more than others. This is because the covariance matrix for each cluster is different. Each boundary curves towards the cluster in the pair that has a smaller spread because the probability density is more highly concentrated for that cluster.

To make the ML classification the same as the nearest-centroid classification, we just need to use the identity matrix instead of a cluster's covariance matrix when calculating the PDF. This will make each distribution uniform in all directions and ensure the shape of the distributions will be the same across all of the classes, meaning only the distance from the centroids will factor into the classification.

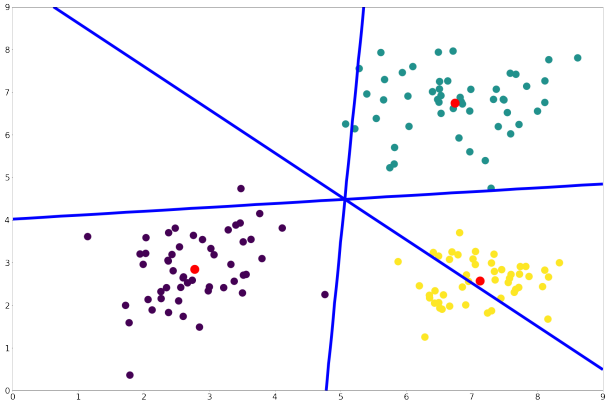

Figure 8: Making maximum-likelihood classifier look like nearest-centroid classifier

If we know one class is more likely than the rest of the classes we can give that class a weighting which changes the decision boundaries in its favour. For example, when one class is twice as likely we need to double the PDF of that class before calculating the pairwise likelihood. The decision boundaries for that class now encompass a larger area, though not a drastically larger one due to the majority of the probability mass being concentrated in the centre of the cluster.

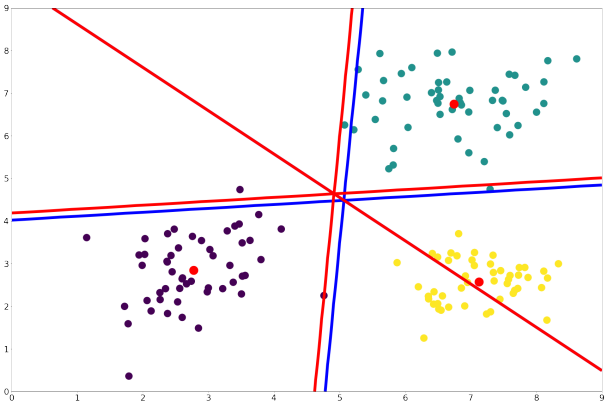

Figure 9: Decision boundaries where one class is twice as likely as the other two

In this case doubling the likelihood of the yellow cluster means that 2 points from the other clusters, which were near to the boundary lines between their clusters and the yellow cluster, would now be put into the yellow class if we ran this classifier on them.

6 Discussion of results

We expected the ML classification to be more accurate than the nearest-centroid classification due it taking into account the spread and shape of the clusters, which means that it'll be better at classifying edge cases.

However, from our results we can see that both of our classifiers separate out the clusters equally well and comfortably classify all of our test data into the classes we would expect them to fit into.

Our expectations for the nearest-centroid classification were exceeded by our observations. This is likely due to our clusters being quite well separated, which makes the difference between the classifications negligible since they only really differ in the edge cases. Our test data didn't contain any edge cases, so it was classified exactly the same by both methods. If the test data was more spread out, there might have been an appreciable difference between the two classifiers.

We also expected our data sets to have some meaningful differences between them when it came to classifying the test data, but we didn't find any. In both cases there were

no differences between the effectiveness of the 2 classification methods.

References

[1] Song Liu. *95-percent Contour Line of 2-dimensional Gaussian Distribution.*