cw1

March 17, 2017

1 COMS21202: Symbols, Patterns and Signals

CW1: *This marked assignment has 3 parts to be undertaken in weeks 17-19 with students working in pairs.*

```
In [11]: import numpy as np
         from sklearn.cluster import KMeans
         from scipy.spatial.distance import cdist
         from scipy.spatial import Voronoi, voronoi_plot_2d
         from scipy import stats
         import matplotlib.pyplot as plt


         %matplotlib inline
         # notebook
         import matplotlib.pylab as pylab
         import matplotlib.mlab as mlab
         pylab.rcParams['figure.figsize'] = (32.0, 24.0)
         pylab.rcParams['font.size'] = 24
```

1.1 Objectives

The goal of this assignment is for you to gain experience with clustering and classifying data. You will be given two sets of data points, each containing values for 5 different attributes (features) derived from an unspecified number of different object classes. One set contains 150 points and you should use it to *train* your classifier. The second set contains 15 points and you should use it to *test* your classifier. A complication is that the training set is **not** labelled - you will need to analyse the feature values, identify two features which are able to separate the different object classes and then cluster the points to obtain class labels which can then be used to train a classifier.

The steps that you should follow are detailed below. You will need to become familiar with several new `Python` commands and so we advise that you make use of the `Python` help facility. You will each be given different data files, with different attribute values, different class parameters and different distinguishing features. Your results will therefore not be the same as that of other students. For those working in pairs you will have to carry out the work for both data sets and be able to explain the differences obtained.

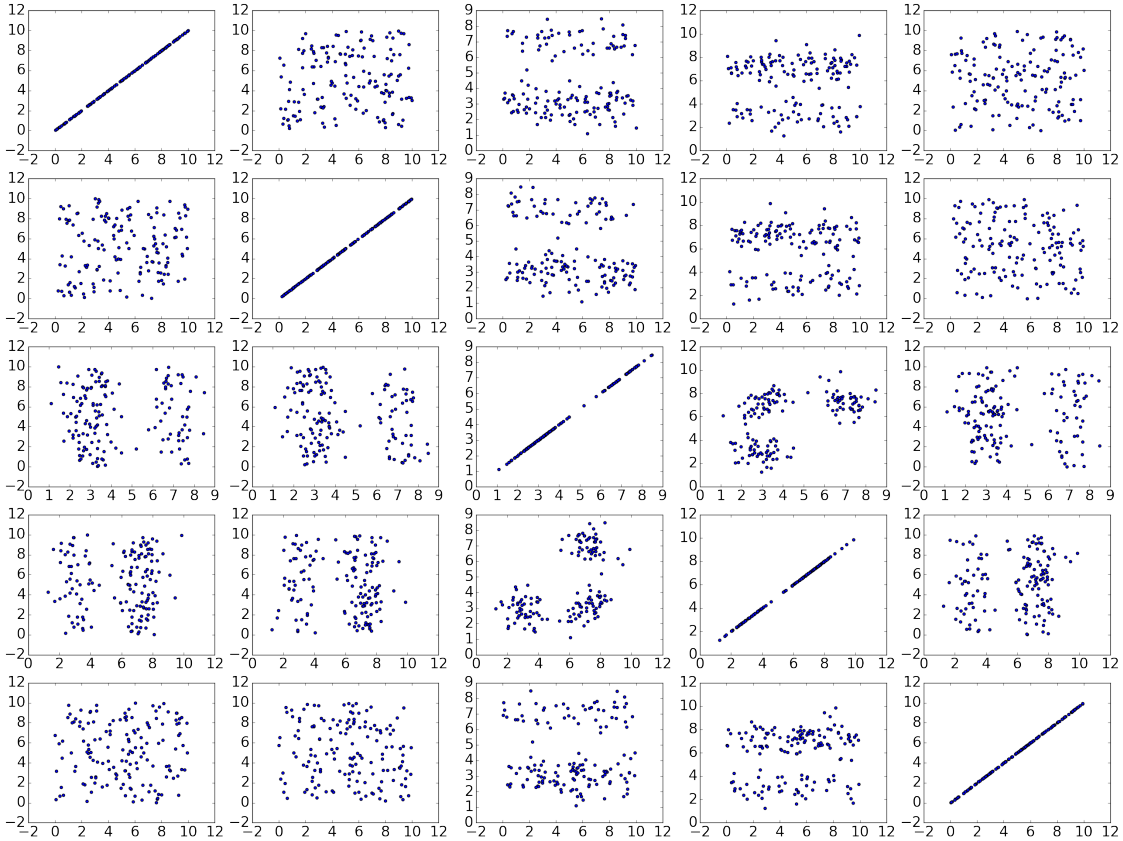
1.2 CW1a (week 17)

1.2.1 1.

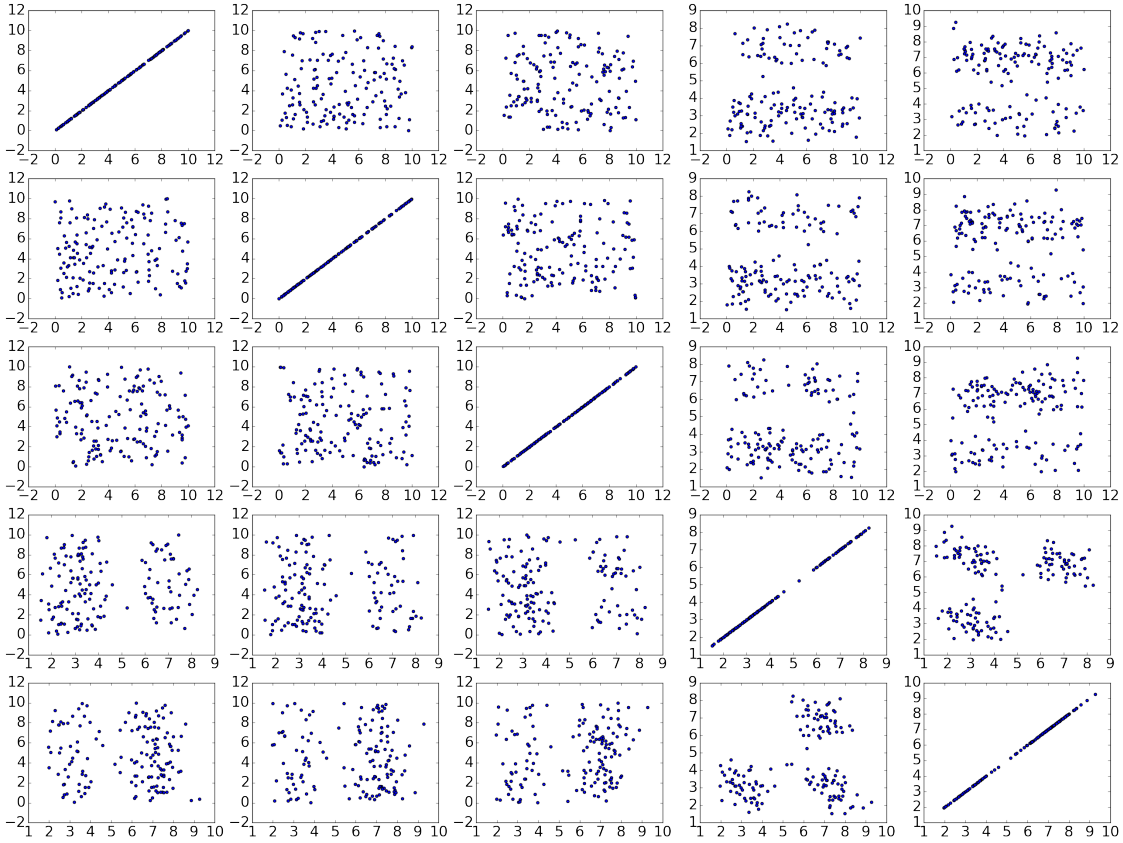
Collect your training and test data from here. You then need to identify which two features separate the classes in your training data. The best way of doing this is to visualise the data by plotting the attribute values for each pair of features. Use the `np.loadtxt` command to read in the data and the plotting commands (try `plotmatrix`) to plot attribute values against each other. Visually inspect the results to understand the data and argue which features best separate the classes. Once you identified the two features create a new data matrix `X` holding only those features as columns.

```
In [2]: def plotmatrix(Matrix):
            r, c = Matrix.shape
            fig = plt.figure()
            plotID = 1
            for i in range(c):
              for j in range(c):
                ax = fig.add_subplot( c, c, plotID )
                ax.scatter( Matrix[:,i], Matrix[:,j] )
                plotID += 1
            plt.show()

        ls_train = np.loadtxt("ls14172.train")
        ls_test = np.loadtxt("ls14172.test")

        es_train = np.loadtxt("es15563.train")
        es_test = np.loadtxt("es15563.test")

        print("Luke's:")
        plotmatrix(ls_train)
        print("\n\nEwan's:")
        plotmatrix(es_train)
```

```
Luke's:
```



Ewan's:



1.2.2 Your comments

For Luke's training data, the most visible clustering is in scatter charts (3,4) and (4,3), and for Ewan's it looks like (4,5) and (5,4) has the clearest clusters. Both of these feature combinations have 3 clusters.

Therefore we will create two new data matrices with only these corresponding data columns, to focus on those features.

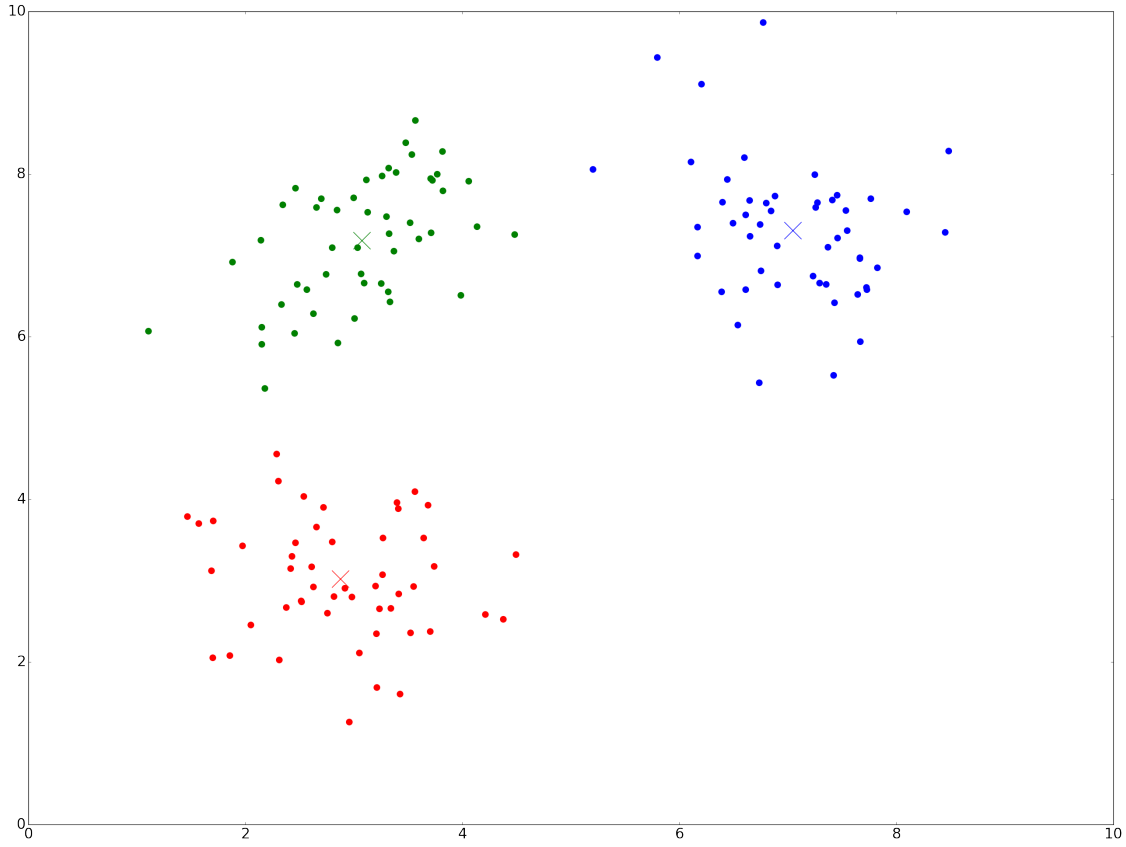
```
In [3]: ls_train_X =  np.matrix(np.column_stack((ls_train[:,2], ls_train[:,3])))
        es_train_X = np.matrix(np.column_stack((es_train[:,3], es_train[:,4])))
```

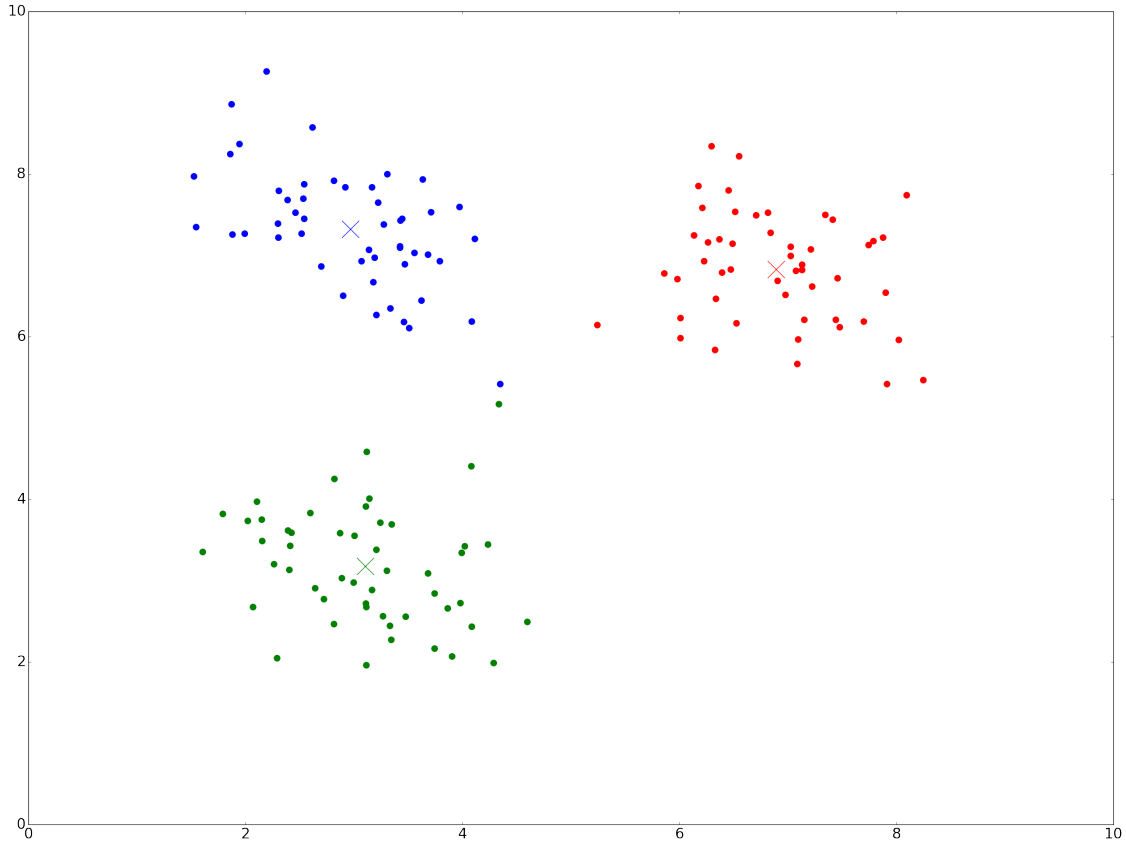
1.2.3 2.

You now need to derive class labels for each of the data points in the training data. You can do this automatically using the K-means algorithm applied to `X`, where K is the number of classes you identified in the previous step. Use the function `kmeans` to do this. Next, you want to visualise the outcome of the K-means clustering. One way to do this is to plot each cluster found by K-means in a different colour. To this end, use the vector of cluster indices returned by `kmeans` to store the points for each class in separate matrices by means of the `np.where` command.

```
In [4]: def kmeans(Data, NClusters):
            km = KMeans(NClusters)
```

```
    fit = km.fit(Data)
    return (fit.cluster_centers_, fit.labels_, fit.inertia_)

# put your code here

colours = ['red', 'green', 'blue', 'purple', 'orange']

def calc_kmeans_groups(X, k):
    centers, labels, _ = kmeans(X, k)
    groups = []
    for i in range(k):
        groups.append(X[np.where(labels==i)])
    return (centers, groups)

def plot_kmeans(X, k):

    centers, groups = calc_kmeans_groups(X, k)

    fig = plt.figure()
    ax = fig.add_subplot(111)

    for i, group in enumerate(groups):
        ax.scatter(group[:,0], group[:,1], color=colours[i], s=100)

    for i in range(k):
        ax.scatter(centers[i][0],centers[i][1],color=colours[i], marker='x'

    ax.set_ylim([0,10])
    ax.set_xlim([0,10])
    plt.show()


plot_kmeans(ls_train_X, 3)
plot_kmeans(es_train_X, 3)
```


10
8
6
4
2
0
0
2
4
6
8
10



1.2.4 Your comments

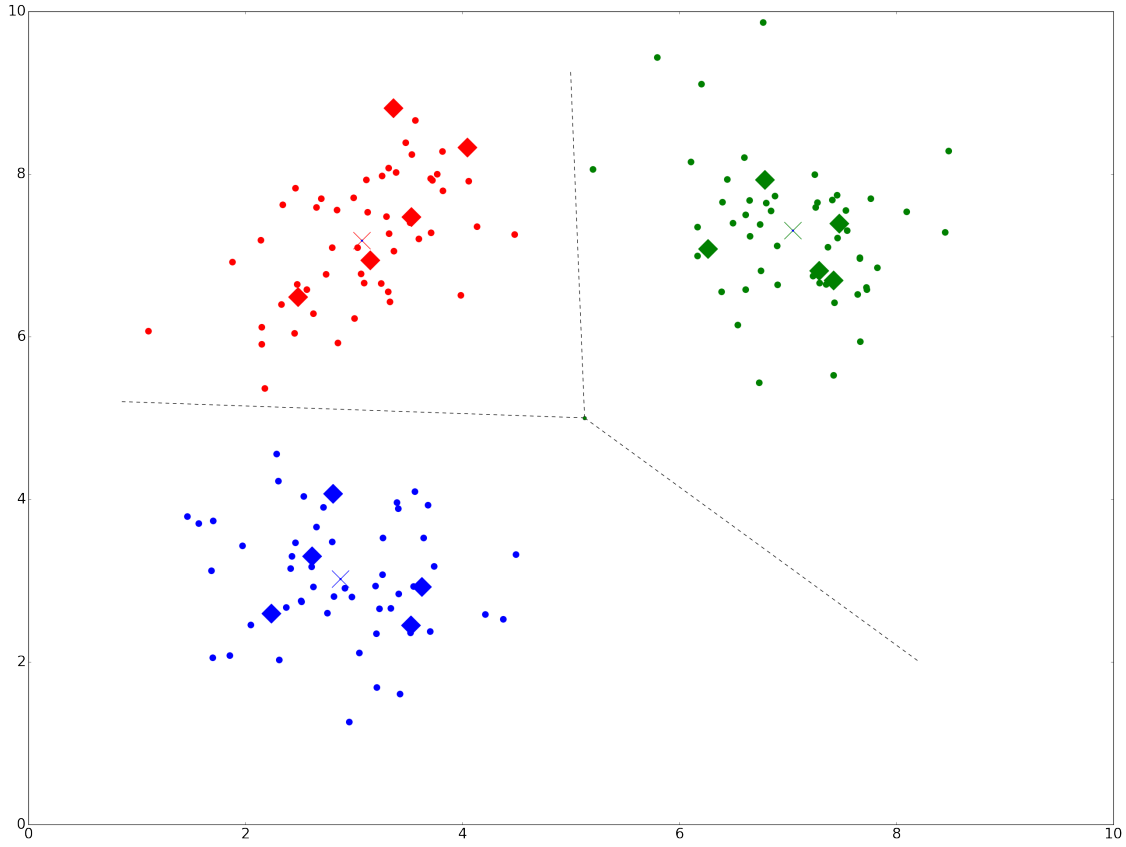
By the colour coding it looks like the KMeans clustering worked to separate the data into three visually distinct classes.

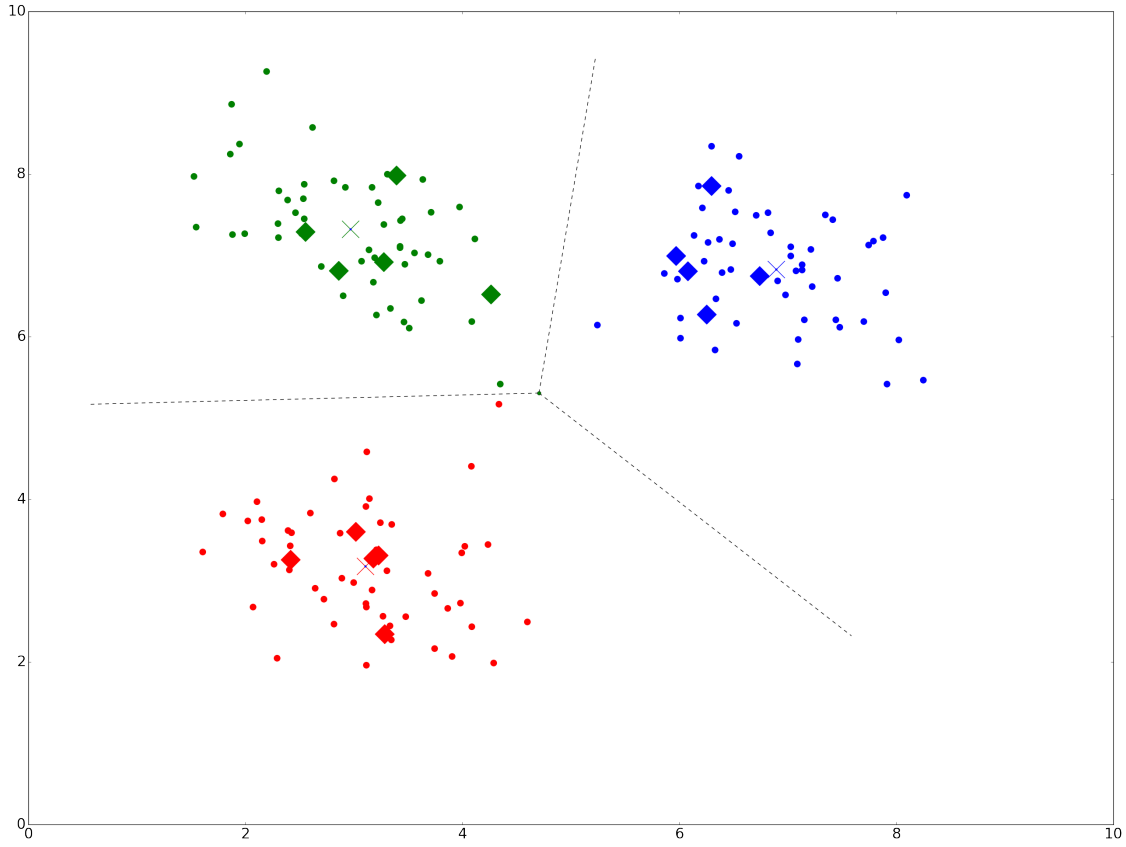
1.2.5 3.

The centroids found by K-means can now be used as a simple nearest-neighbour classifier. Load your test data and select the two relevant features. Using a combination of the `cdist(X, metric='euclidean', p=2)` and `np.argmin` (with `axis` parameter) commands, construct a vector of labels for each of the test points indicating which cluster centroid is the nearest. Plot the test points in the previous plot with the colour indicating their class but a different symbol. The corresponding decision boundaries can be plotted by passing the centroids to the `Voronoi` and `voronoi_plot_2d` functions (remember to set the `ax` parameter of the later one to your current plot).

```
In [30]: ls_test_X = np.matrix(np.column_stack((ls_test[:,2], ls_test[:,3])))
         es_test_X = np.matrix(np.column_stack((es_test[:,3], es_test[:,4])))


         def calc_nearest_neighbor(test_X, centers, k):
```

```
    labels = np.argmin(cdist(centers, test_X, metric='euclidean', p=2), ax
    groups = []
    for i in range(k):
        groups.append(test_X[np.where(labels==i)])
    return(groups)


def plot_nearest_neighbor(test_X, train_X, k):
    centers, train_groups = calc_kmeans_groups(train_X, k)
    test_groups = calc_nearest_neighbor(test_X, centers, k)
    fig = plt.figure()
    ax = fig.add_subplot(111)
    for i in range(k):
        ax.scatter(centers[i][0], centers[i][1], color=colours[i], marker=
        ax.scatter(train_groups[i][:,0], train_groups[i][:,1], color=colou
        ax.scatter(test_groups[i][:,0], test_groups[i][:,1], color=colours

    vor = Voronoi(centers)
    voronoi_plot_2d(vor, ax=ax, linewidths=4)

    ax.set_ylim([0,10])
    ax.set_xlim([0,10])

    plt.show()


plot_nearest_neighbor(ls_test_X, ls_train_X, 3)
plot_nearest_neighbor(es_test_X, es_train_X, 3)
```


0
2
4
6
8
10
0
2
4
6
8
10



1.2.6 Your comments

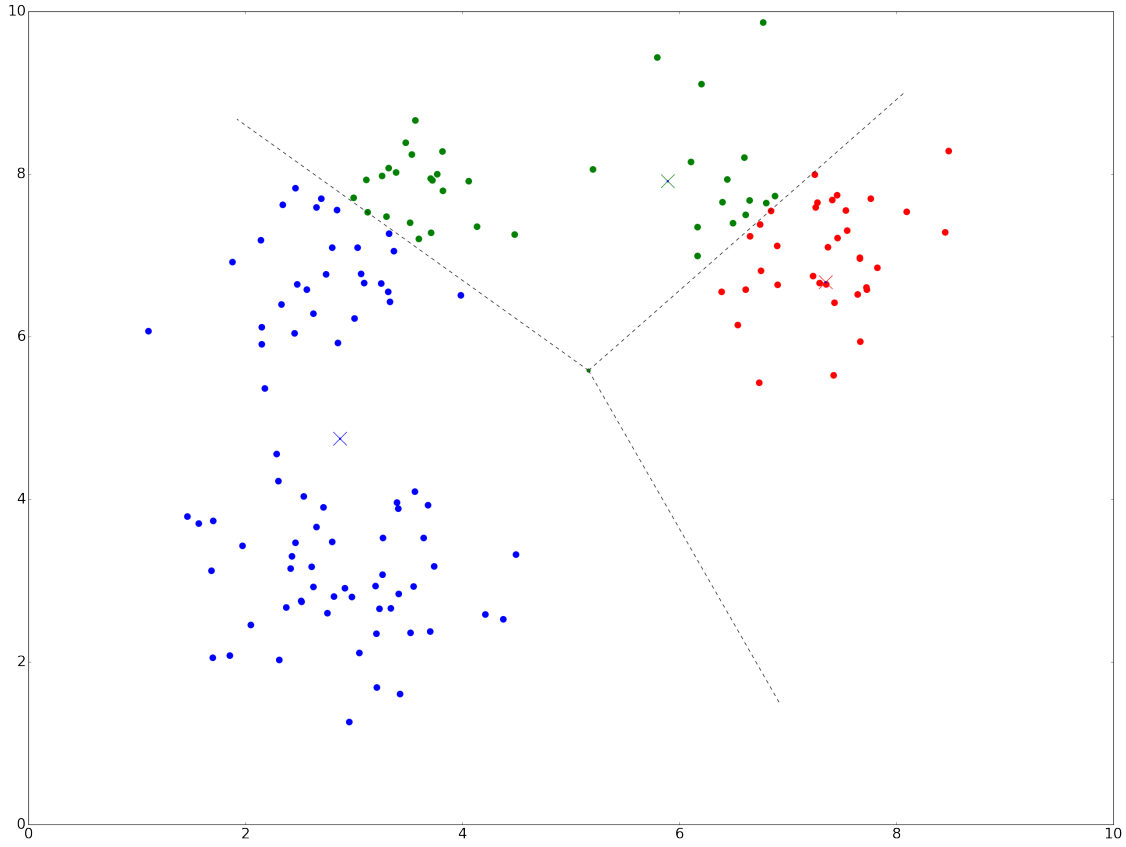
Both look good, accurate clustering and Voronoi lines, note Ewan's two training data points close to the boundary points. All test data classification matches their visual cluster

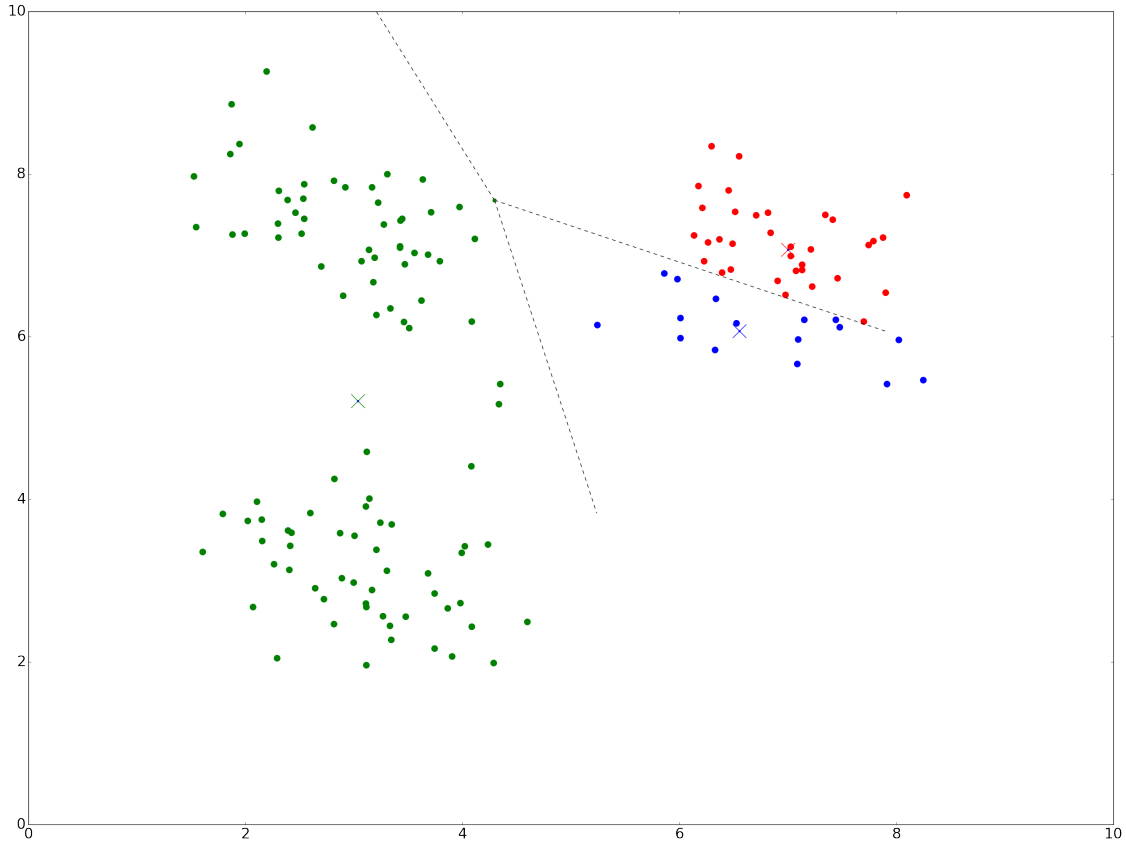
1.2.7 4.

Finally, we want you to deliberately find a non-optimal clustering. To this end, use the within-cluster sums of point-to-centroid distances returned by K-means to keep calling K-means until you find a clearly non-optimal value. You may need to change the definition of `kmeans` function and play around with its initialisation parameters. Plot the Voronoi diagram in the same plot to demonstrate non-optimality.

```
In [27]: def kmeans_suboptimal(Data, NClusters, runs=100):
             worst_inertia = 0
             worst_fit = None
             for i in range(runs):
                 km = KMeans(NClusters, init='random', n_init=1, max_iter=5)
                 this_fit = km.fit(Data)
                 if this_fit.inertia_ > worst_inertia:
                     worst_fit = this_fit
```

```
            worst_inertia = this_fit.inertia_
    return (worst_fit.cluster_centers_, worst_fit.labels_, worst_fit.inert

def calc_kmeans_groups_suboptimal(X, k):
    centers, labels, _ = kmeans_suboptimal(X, k)
    groups = []
    for i in range(k):
        groups.append(X[np.where(labels==i)])
    return (centers, groups)

def plot_kmeans_suboptimal(train_X, k):
    centers, groups = calc_kmeans_groups_suboptimal(train_X, k)
    fig = plt.figure()
    ax = fig.add_subplot(111)
    for i in range(k):
        ax.scatter(centers[i][0], centers[i][1], color=colours[i], marker=
        ax.scatter(groups[i][:,0], groups[i][:,1], color=colours[i], s=100

    vor = Voronoi(centers)
    voronoi_plot_2d(vor, ax=ax, linewidths=4)

    ax.set_ylim([0,10])
    ax.set_xlim([0,10])

    plt.show()


plot_kmeans_suboptimal(ls_train_X, 3)
plot_kmeans_suboptimal(es_train_X, 3)
```


0
2
4
6
8
10
0
2
4
6
8
10



1.2.8 Your comments

The default Kmeans function runs the algoithm 10 times, then picks the fit with the least inertia. We reduced this n_init parameter from 10 to 1, then ran it 100 times to find the solution with the worst inertia.

Although Euan's data could be suboptimally clustered by this method, for Luke's data to also be suboptimally clustered, the max_iter parameter (iterations of the kmeans algorithm per run) also had to be lowered, from 300 to 5.

1.3 CW1b (week 18)

The simple nearest-centroid classifier you constructed in the previous assignment leads to linear decision boundaries between each pair of classes, as visualised by the Voronoi diagram. We now want you to contrast and compare this with a maximum-likelihood classifier.

1.3.1 1.

Model the data in each class as being generated from a 2-D Normal Distribution. Estimate the class means and covariances from the labelled training data using the `np.mean` (remember about `axis` parameter) and `np.cov` functions. Visualise the estimated distributions by plotting contours of the `stats.multivariate_normal.pdf` functions of each class. Choose the contour level such

that 95% of the probability mass is within the ellipse. Use as a starting point that for bivariate random distributions the points on this contour satisfy the equation

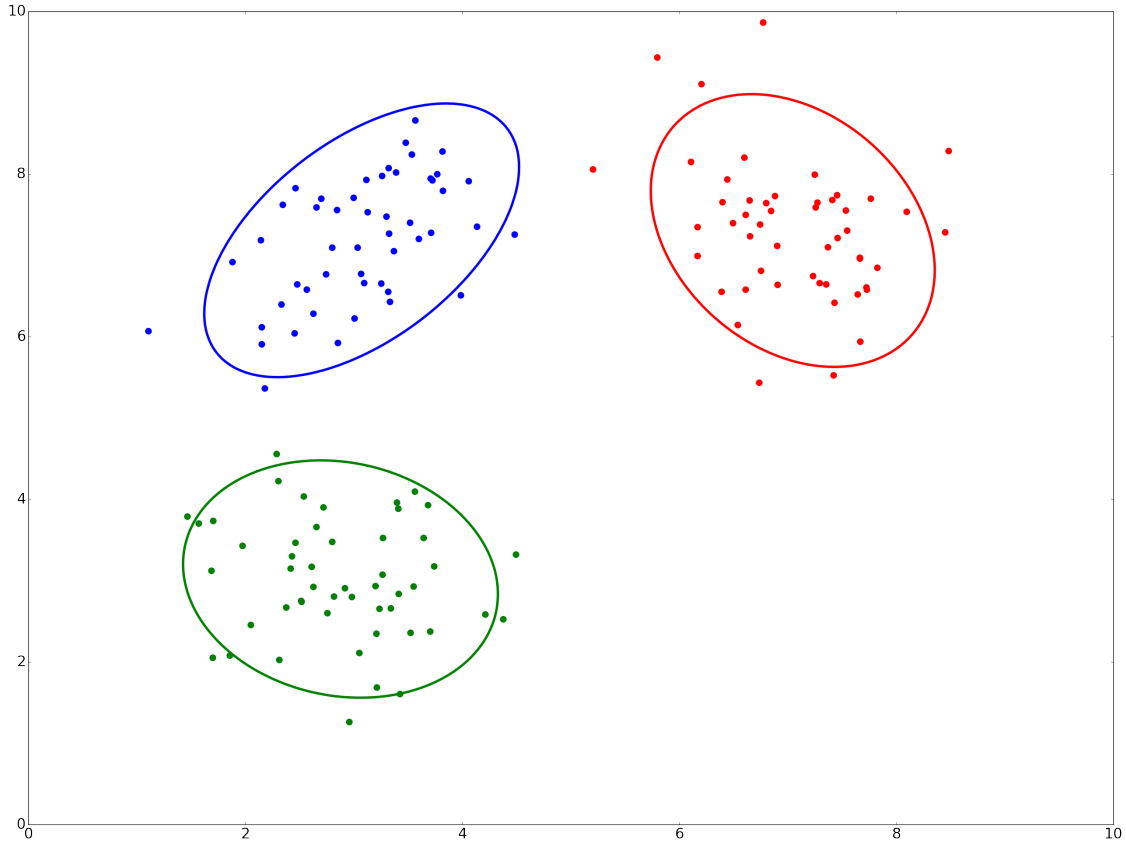
$$(x-\mu)^T * \Sigma^{-1} * (x-\mu)/2 = 3$$

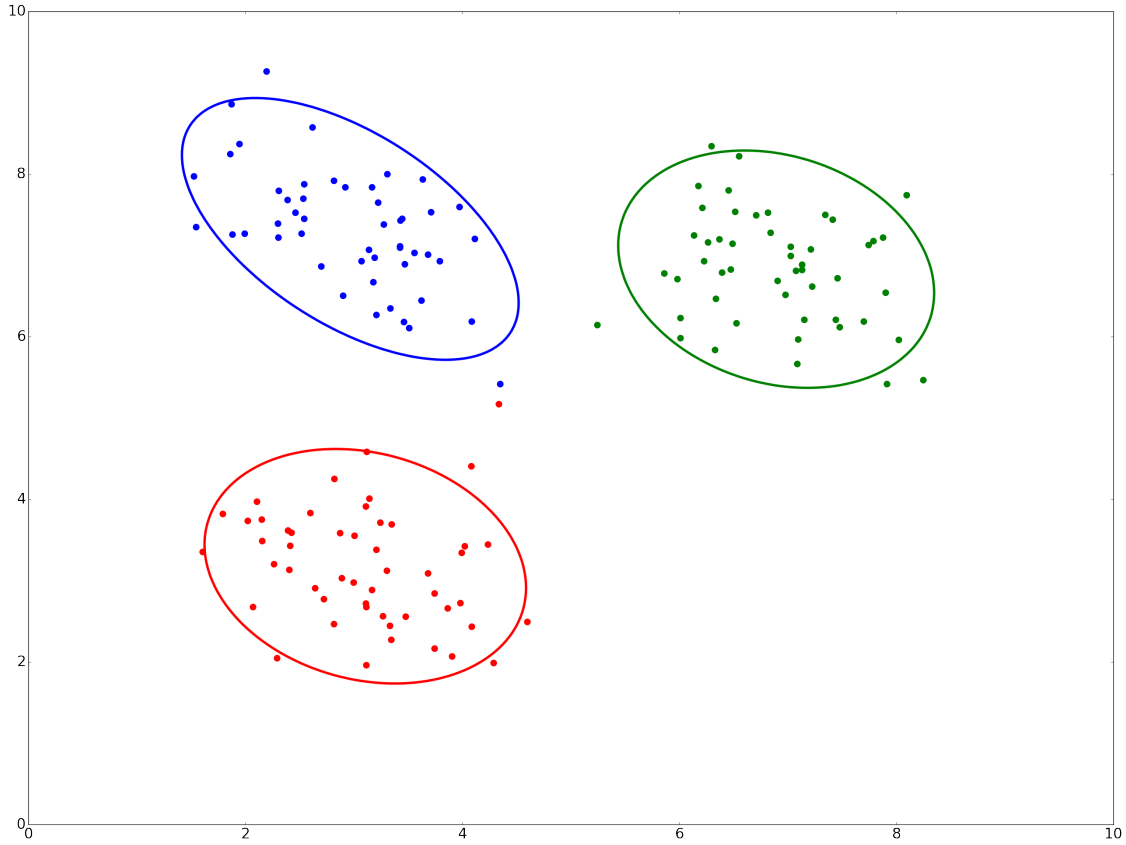
and use this to find the appropriate density for the contour level. `Python` functions to use include `plt.contour` and `np.meshgrid`.

Clarification: this basically tells you to draw an ellipse at squared Mahalanobis distance 6. Why 6 (actually 5.99)? This comes from the fact that sums of squared Gaussians follow a Chi-squared distribution, so we use the inverse of the Chi-squared cumulative distribution with 2 degrees of freedom. You can verify this in `Python` by the query `stats.chi2.ppf(0.95, 2)` which gives 5.9915. This is the 2-D equivalent of saying that in 1-D, 95% of the Gaussian probability mass is within +/- 2 standard deviations from the mean (actually 1.96). More information at the following links:
* http://www.visiondummy.com/2014/04/draw-error-ellipse-representing-covariance-mat
* http://www.cs.ubc.ca/~murphyk/Teaching/CS340-Fall06/reading/gaussians.pdf

In [26]:
```
def calc_normal_paras(groups):
    mus= []
    covars = []
    for group in groups:
        mus.append((np.mean(group, axis=0).tolist()[0]))
        covars.append(np.cov(group, rowvar=0).tolist())
    return (mus, covars)



def calc_contour_levels(groups, k, identity_covars=False, bias_class=None)
    mus, covars = calc_normal_paras(groups)
    if identity_covars:
        for i,_ in enumerate(covars):
            covars[i] = np.identity(len(covars[i]))

    if bias_class is not None:
        covars[bias_class] = np.multiply(covars[bias_class], 2)

    ##Using example from the documentation: https://docs.scipy.org/doc/sci
    #Grid points for manipulating into contour
    X, Y = np.meshgrid(np.linspace(0, 10, 200), np.linspace(0, 10, 200))

    #Store grid positions in pos
    pos = np.empty(X.shape + (2,))
    pos[:,:,0] = X
    pos[:,:,1] = Y


    probs = []
    contours = []
```

```
    for i in range(k):
        #Use mus and covars to generate a multivariate normal distributio
        probs.append(stats.multivariate_normal(mus[i], covars[i]).pdf(pos)

        #substitution of the given equation from the question resolves to
        contours.append((1/2.0 * np.pi * np.sqrt(np.linalg.det(covars[i]))
    return (X, Y, probs, contours)


def plot_contours(data, k):
    fig = plt.figure()
    ax = fig.add_subplot(111)
    _, groups = calc_kmeans_groups(data, k)

    X, Y, probs, contours = calc_contour_levels(groups, k)
    for i in range(k):
        x = (groups[i][:,0])
        y = (groups[i][:,1])


        ax.scatter(x, y, color = colours[i], s=100)
        ax.contour(X, Y, probs[i], [contours[i]], colors = colours[i], lin

    ax.set_ylim([0,10])
    ax.set_xlim([0,10])
    plt.show()


plot_contours(ls_train_X,3)
plot_contours(es_train_X,3)
```


0
2
4
6
8
10
0
2
4
6
8
10



1.3.2 Your comments

not 95% of points, but according to a normal distribution, so looks ok

1.3.3 2.

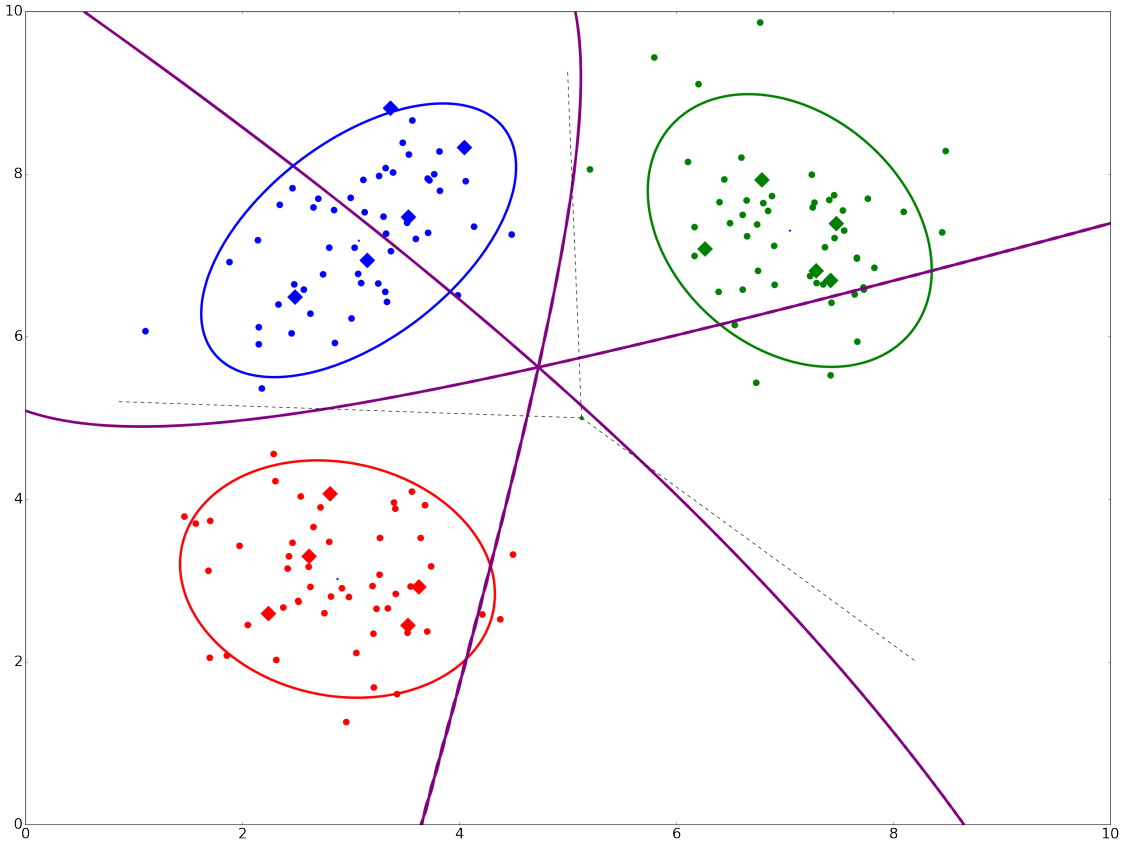
Now plot the decision boundaries between each pair of classes. This can be done once again using the `plt.contour` function, this time on each of the three pairwise likelihood ratios. Visually inspect the decision boundaries and make sure you understand why they have a particular shape. Also check whether this maximum-likelihood classifier disagrees with the nearest-centroid classifier on any of the test or training points (for both your data sets if you're working in pairs).

```
In [33]: def ml_classify(mus, covars, test_point):
             # Maximum Likelihood Estimate (MLE):
             #choose class C that maximizes probability of test data: P(x|C)
             # C' = argmax_C (P(x|C))
             # x belongs to class C' which is the class with the maximum P(x|C)
             probs = []
             for i in range(0,len(mus)):
                 probs.append([i,stats.multivariate_normal(mus[i], covars[i]).pdf(t
             probs = np.asarray(probs)
```

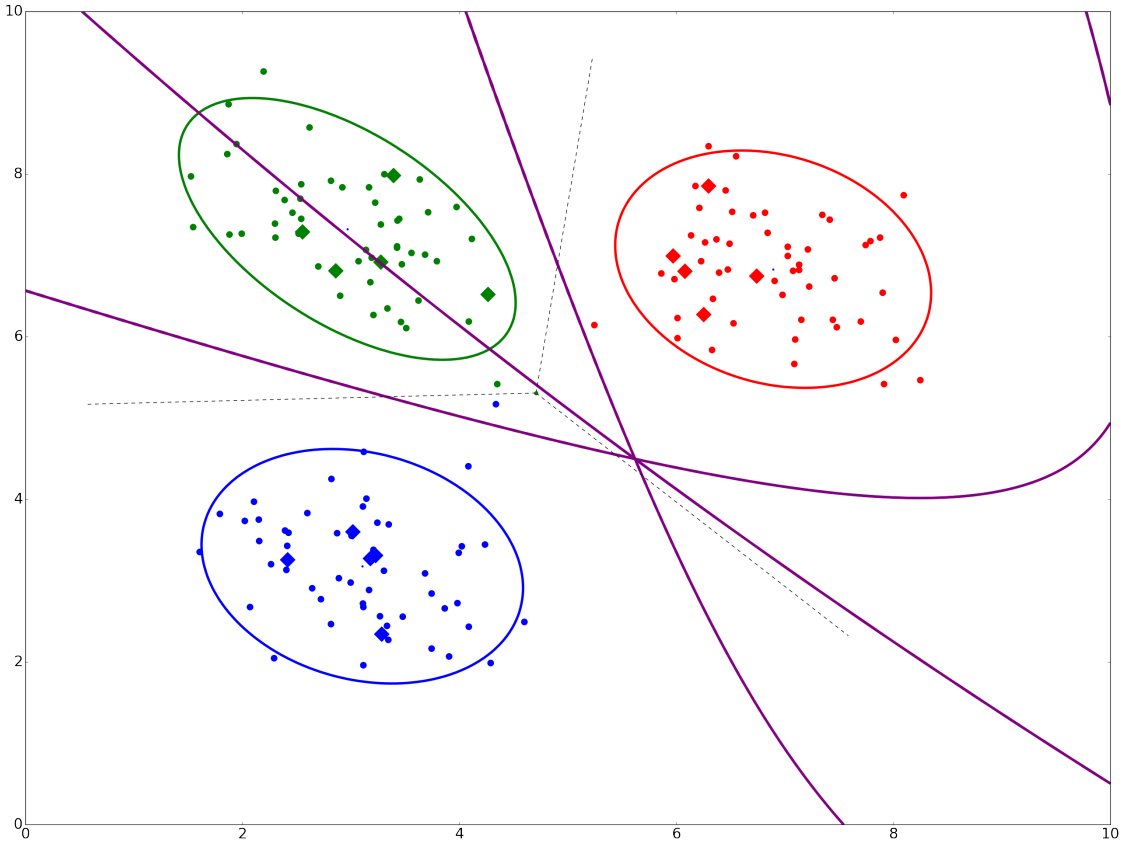
```
    max_prob = probs[:,1].argmax()
    #get max index
    return max_prob

def find_index(array, vec):
    for i, row in enumerate(array):
        if vec in row:
            return i
    return -1

def plot_contours_and_boundaries(train_X, test_X, k, check_test_class=Fals
    fig = plt.figure()
    ax = fig.add_subplot(111)
    centers, train_groups = calc_kmeans_groups(train_X, k)
    # for comparison:
    test_groups = calc_nearest_neighbor(test_X, centers, k)

    X, Y, probs, contours = calc_contour_levels(train_groups, k, identity_

    for i in range(k):
        x = (train_groups[i][:,0])
        y = (train_groups[i][:,1])

        ax.scatter(x, y, color = colours[i], s=100)
        ax.contour(X, Y, probs[i], [contours[i]], colors = colours[i], lin
        for j in range(k):
            ax.contour(X, Y, np.divide(probs[i], probs[j]), [1], colors =

    if check_test_class:
        for i, point in enumerate(test_X):
            mus, covars = calc_normal_paras(train_groups)
            ml_class = ml_classify(mus, covars, point)

            ax.scatter(point.item(0), point.item(1), color = colours[ml_cl
            nc_class = find_index(test_groups, point)

            diff = "Match: " if (ml_class==nc_class) else "Mismatch: "
            print (diff + "Test point " + str(i) + " has MLE " + str(ml_cl



    if bias_class is not None:
        X, Y, probs, contours = calc_contour_levels(train_groups, k, ident
        ax.contour(X, Y, probs[bias_class], [contours[bias_class]], colors
        for j in range(k):
            ax.contour(X, Y, np.divide(probs[bias_class], probs[j]), [1],
    else:
        vor = Voronoi(centers)
```

```
                voronoi_plot_2d(vor, ax=ax)


            ax.set_ylim([0,10])
            ax.set_xlim([0,10])
            plt.show()


        print ("Luke's Data:")
        plot_contours_and_boundaries(ls_train_X, ls_test_X, 3, check_test_class=Tr
        print ("Ewan's Data:")
        plot_contours_and_boundaries(es_train_X, es_test_X, 3, check_test_class=Tr
```

```
Luke's Data:
Match: Test point 0 has MLE 1 and Nearest Centroid 1
Match: Test point 1 has MLE 1 and Nearest Centroid 1
Match: Test point 2 has MLE 1 and Nearest Centroid 1
Match: Test point 3 has MLE 2 and Nearest Centroid 2
Match: Test point 4 has MLE 0 and Nearest Centroid 0
Match: Test point 5 has MLE 1 and Nearest Centroid 1
Match: Test point 6 has MLE 2 and Nearest Centroid 2
Match: Test point 7 has MLE 0 and Nearest Centroid 0
Match: Test point 8 has MLE 1 and Nearest Centroid 1
Match: Test point 9 has MLE 0 and Nearest Centroid 0
Match: Test point 10 has MLE 2 and Nearest Centroid 2
Match: Test point 11 has MLE 0 and Nearest Centroid 0
Match: Test point 12 has MLE 0 and Nearest Centroid 0
Match: Test point 13 has MLE 2 and Nearest Centroid 2
Match: Test point 14 has MLE 2 and Nearest Centroid 2
```




```
Ewan's Data:
Match: Test point 0 has MLE 2 and Nearest Centroid 2
Match: Test point 1 has MLE 1 and Nearest Centroid 1
Match: Test point 2 has MLE 1 and Nearest Centroid 1
Match: Test point 3 has MLE 0 and Nearest Centroid 0
Match: Test point 4 has MLE 1 and Nearest Centroid 1
Match: Test point 5 has MLE 2 and Nearest Centroid 2
Match: Test point 6 has MLE 0 and Nearest Centroid 0
Match: Test point 7 has MLE 0 and Nearest Centroid 0
Match: Test point 8 has MLE 1 and Nearest Centroid 1
Match: Test point 9 has MLE 2 and Nearest Centroid 2
Match: Test point 10 has MLE 2 and Nearest Centroid 2
Match: Test point 11 has MLE 0 and Nearest Centroid 0
Match: Test point 12 has MLE 2 and Nearest Centroid 2
Match: Test point 13 has MLE 1 and Nearest Centroid 1
Match: Test point 14 has MLE 0 and Nearest Centroid 0
```



1.3.4 Your comments

visually, all test points in the same boundary, also printed all ok. Some training points in different class though.

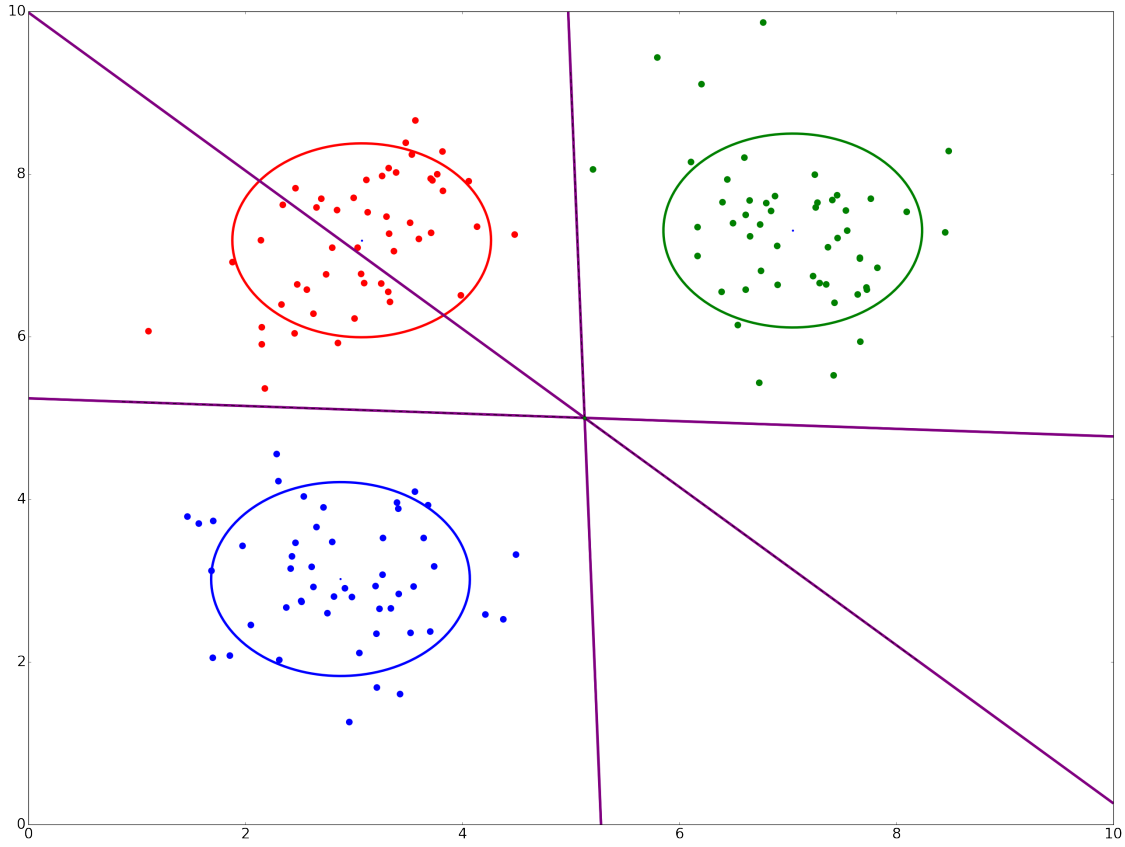
contour boundaries look good, ideally wouldnt be drawn past the intersection but limited with the contour function.

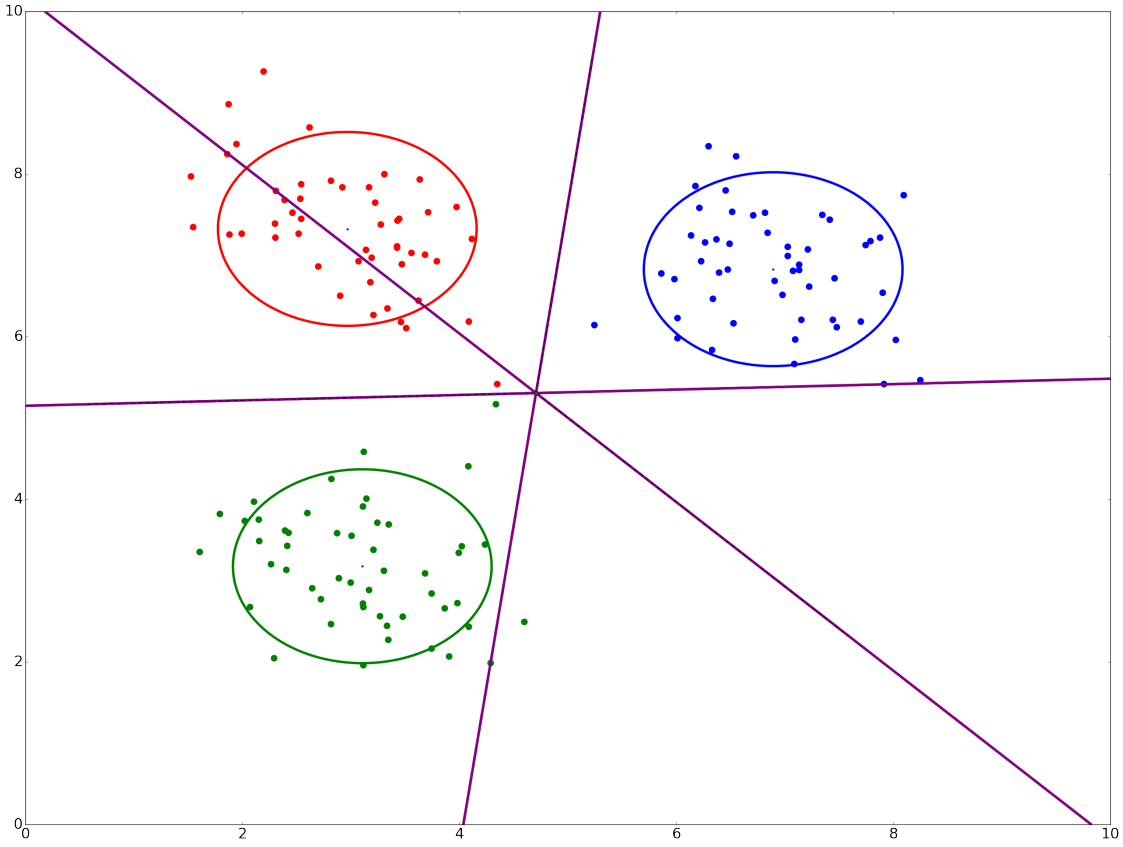
diamonds are the MLE classifications, and we've plotted the nearest centroid voronoi lines too. The printed log shows the classification given to each point in the test data and that the classification of both methods matches. The only main visible difference is that one of the training points for the lower-left class on Ewan's data lies on the other side of the decision boundary, whereas in the nearest centroid classification all training points where within their respective voronoi boundaries.

1.3.5 3.

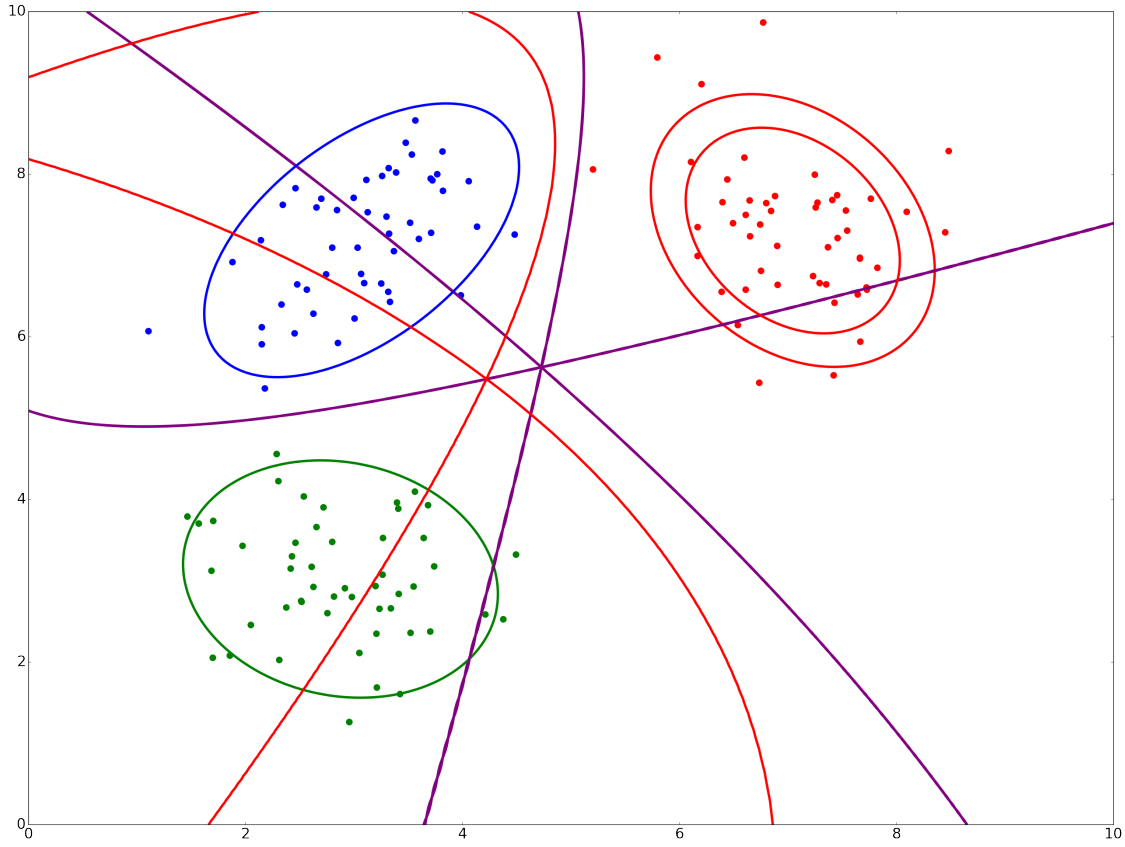
How would you have to change the maximum-likelihood classifier so that its decision boundaries are the same as the ones for nearest-centroid? And how would you change it if you know that one of the three classes is twice as likely as the other two? Demonstrate the effect graphically.

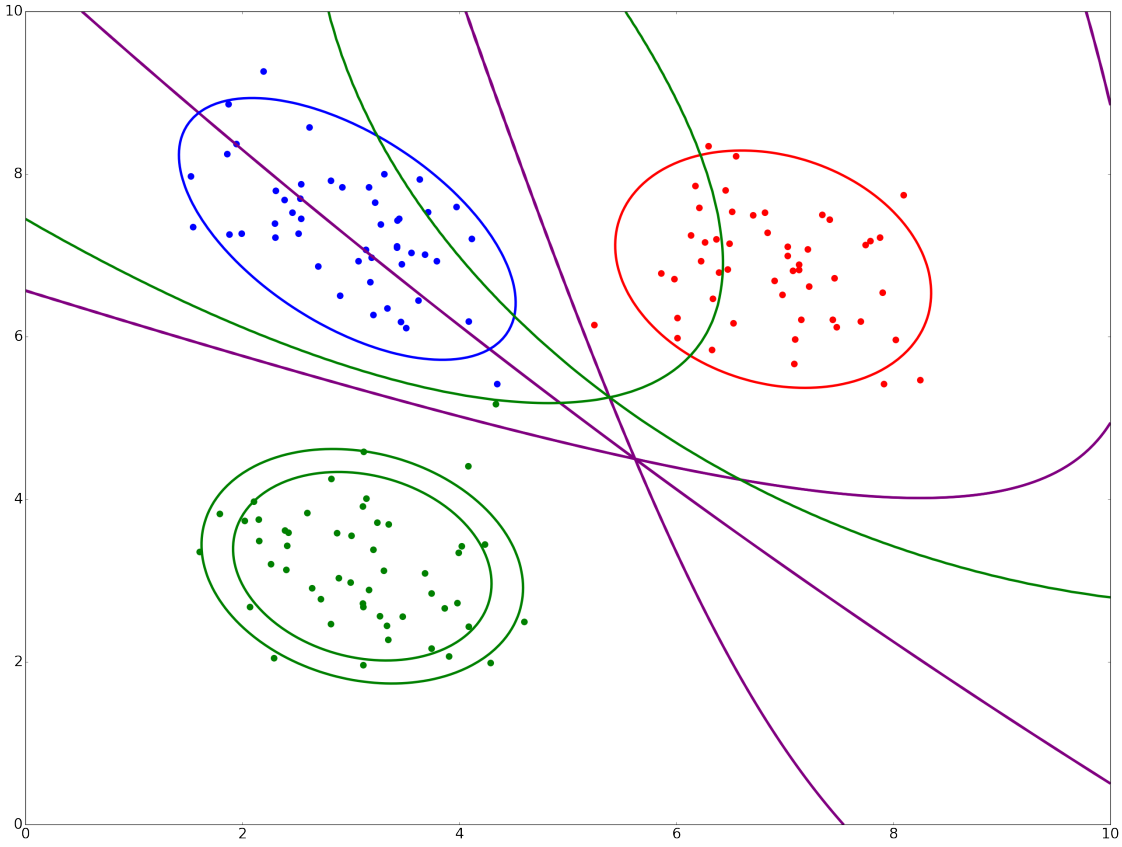
```
In [35]: plot_contours_and_boundaries(ls_train_X, ls_test_X, 3, identity_covars=Tru
         plot_contours_and_boundaries(es_train_X, es_test_X, 3, identity_covars=Tru
```


10
8
6
4
2
0
0
2
4
6
8
10




```
In [36]: plot_contours_and_boundaries(ls_train_X, ls_test_X, 3, bias_class=0)
         plot_contours_and_boundaries(es_train_X, es_test_X, 3, bias_class=1)
```


0
2
4
6
8
10
0
2
4
6
8
10



purple is original, then the new contour and new classification lines are int he biased class' colour

1.3.6 Your comments

Put here any comments

1.4 CW1c (week 19)

You now need to write a report which describes, explains and analyses the work you have done for **CW1a** and **CW1b**. The aim of this report is to demonstrate your understanding of methods you used and the results that you have obtained. This will be useful training for the CW2 assignment which is entirely assessed through a report.

As this is not a `Python` programming assignment you are not allowed to use any `Python` code in your report. So, rather than "The program then calls `kmeans(data,3)` and stores the result in `some_var`" you should write something like "we then applied K-means clustering to this data set to obtain K=3 clusters".

The report should be **no more than 4 pages long using no less than 11 point font** and should be submitted in `pdf` format on SAFE. You will continue to work in pairs and at least one of you should upload the report. Make sure that the submitted report clearly identifies the authors.

We suggest you use the following headings in the report (but this is not prescriptive): 1. Introduction 2. Feature selection 3. Identifying the classes 4. Nearest-centroid classification 5. Maximum-likelihood classification 6. Discussion of results 7. Sources used

Notice how these sections don't map one-to-one to the questions above, but rather concentrate on telling a 'story'. In each section you should briefly explain what the question is, how you approached it and what the results are. We will expect you to include plots generated by `Python`, but these should not take more than about *one-third of the report*. Each plot should have a numbered caption with a succinct description. You need to include plots for both datasets assigned to you and your partner.

To gain high marks your report will need to demonstrate a good understanding of the tasks and the methods used, backed up by a clear explanation of your experimental results.