Statistical Patter Recognition
Homework #03
Deadline: Dec 15

By Hesam Mousavi (9931155) and AmirHosein Batouei (40032723)

Bayesian Classification

GDA vs logistic regression is simpler and more efficiency algorithm and sometime works better on small datasets

$$P(y|\mathbf{X}) = \frac{\overbrace{P(\mathbf{X}|y)}^{Likelihood}\overbrace{P(y)}^{Prior}}{\underbrace{P(\mathbf{X})}_{normalizing\ \ factor}}$$

Based on Bayesian Learning and use Bayesian rule

P(X|y) is :

$$x|y=0 \sim \mathcal{N}(\mu_0, \Sigma) \iff p(x|y=0) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_0)^T \Sigma^{-1}(x-\mu_0)\right)$$

$$x|y=1 \sim \mathcal{N}(\mu_1, \Sigma) \iff p(x|y=1) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_1)^T \Sigma^{-1}(x-\mu_1)\right).$$

We know the class label and want to find class distribution within class (fit class distribution) then in test phase find $y^{new} = argmax_y \ P(y|\mathbf{X})$ and as P(X) is the same, we can omit it and just find likelihood and prior

$$P(\mathbf{X}|y=i) = \frac{1}{(\sqrt{2\pi})^n|\Sigma|^{\frac{1}{2}}} \ exp(\frac{-1}{2}(\mathbf{X}-\boldsymbol{\mu}_i)^T\Sigma^{-1}(\mathbf{X}-\boldsymbol{\mu}_i))$$

$$P(y) = \phi_1^{1\{y=1\}}\phi_2^{1\{y=2\}}\cdots\phi_c^{1\{y=c\}}$$

So our parameter $\boldsymbol{\theta} = \{\phi_1, \phi_2, ..., \phi_c, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, ..., \boldsymbol{\mu}_c, \Sigma\}$

As we consider samples are I.I.D. so we our cost function(log-likelihood) is

$$l(\boldsymbol{\theta}) = log \prod_{j=1}^{m} P(\mathbf{X}^{(j)}, y^{(j)}) = log \prod_{j=1}^{m} P(\mathbf{X}^{(j)} | P(y^{(j)})) P(y^{(j)})$$

And to find our parameters

$$\phi_i^{MLE} = \frac{\sum_{j=1}^{m} 1\{y^{(j)} = i\}}{m}$$

$$\boldsymbol{\mu}_i^{MLE} = \frac{\sum_{j=1}^{m} 1\{y^{(j)} = i\} \mathbf{X}^{(j)}}{\sum_{j=1}^{m} 1\{y^{(j)} = i\}}$$

$$\Sigma^{MLE} = \frac{1}{m} \sum_{j=1}^{m} (\mathbf{X}^{(j)} - \boldsymbol{\mu}_{y^{(j)}})(\mathbf{X}^{(j)} - \boldsymbol{\mu}_{y^{(j)}})^T$$

```
    def find_mean_class_i(self, label):
        '''
        Get label and find mean features of samples that has our label
        '''
        where_label = np.where(self.y_train == label)[0]
        sample_label = self.X_train[where_label]
        mean_sample_label = np.mean(sample_label, axis=0).reshape(-1, 1)
        return mean_sample_label

    def find_mean_classes(self):
        '''
        Find mean features of all classes
        '''
        return np.array(
            [self.find_mean_class_i(label).T for label in self.diffrent_label])

    def find_sigma_LDA(self):
        '''
        find sigma
        '''
        X_demean_in_class = np.copy(self.X_train)
        for label in self.diffrent_label:
            where_label = np.where(self.y_train == label)[0]
            X_demean_in_class[where_label] -= self.mean_class[label]
        return np.cov(X_demean_in_class.T)

    def find_parameters_LDA(self):
        self.phi = (
            self.count_diffrent_label_train / self.number_of_sample_train)
        self.mean_class = self.find_mean_classes()
        self.sigma = self.find_sigma_LDA()
```

$$
\begin{aligned}
y^{new} &= argmax_y \; P(y|\mathbf{X}) \\
&= argmax_y \; \frac{P(\mathbf{X}|y)P(y)}{P(X)} \\
&= argmax_y \; P(\mathbf{X}|y)P(y)
\end{aligned}
$$

```
    def find_probabilities_LDA(self, X_input):
        number_of_sample = X_input.shape[0]
        probabilities = np.zeros((number_of_sample, self.number_of_class))
        coefficient = 1  # It's constant and doesn't change argmax
        sigma_inverse = np.linalg.inv(self.sigma)
        for label in self.diffrent_label:
            X_demean = X_input - self.mean_class[label]
            P_X_given_y = coefficient * -0.5 * (
                    ((X_demean @ sigma_inverse) * X_demean)
                    @ np.ones((self.number_of_feature, 1)))
            prior = self.phi[label]
            probabilities[:, label:label+1] = P_X_given_y + np.log(prior)
        return probabilities

    def find_prediction(self, probabilities):
        return np.argmax(probabilities, axis=1).reshape(-1, 1)
```

```
def accuracy_score(y_input, predicted):
    return np.mean(y_input == predicted)


def precision_score(y_input, predicted):
    return (y_input * predicted).sum() / predicted.sum()


def recall_score(y_input, predicted):
    return (y_input * predicted).sum() / y_input.sum()


def f1_score(y_input, predicted):
    return 2 * (y_input * predicted).sum() / (y_input + predicted).sum()
```

In dataset BC-1 accuracy train is 0.9899874843554443 and accuracy test is 1.0
In dataset BC-1 precision train is 0.9899749373433584 and precision test is 1.0
In dataset BC-1 recall train is 0.9899749373433584 and recall test is 1.0
In dataset BC-1 f1 train is 0.9899749373433584 and f1 test is 1.0
In dataset BC-2 accuracy train is 0.9912390488110138 and accuracy test is 1.0
In dataset BC-2 precision train is 1.0 and precision test is 1.0
In dataset BC-2 recall train is 0.9824561403508771 and recall test is 1.0
In dataset BC-2 f1 train is 0.9911504424778761 and f1 test is 1.0

$$\underbrace{\mathbf{x}^T\Sigma^{-1}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)}_{a\mathbf{x}} \overset{-}{\cancel{+}} \underbrace{\frac{1}{2}\boldsymbol{\mu}_i^T\Sigma^{-1}\boldsymbol{\mu}_i \overset{+}{\cancel{-}} \frac{1}{2}\boldsymbol{\mu}_j^T\Sigma^{-1}\boldsymbol{\mu}_j + log\ \frac{P(y=i)}{P(y=j)}}_{b} = 0$$

In LDA:

$ax + b = 0$

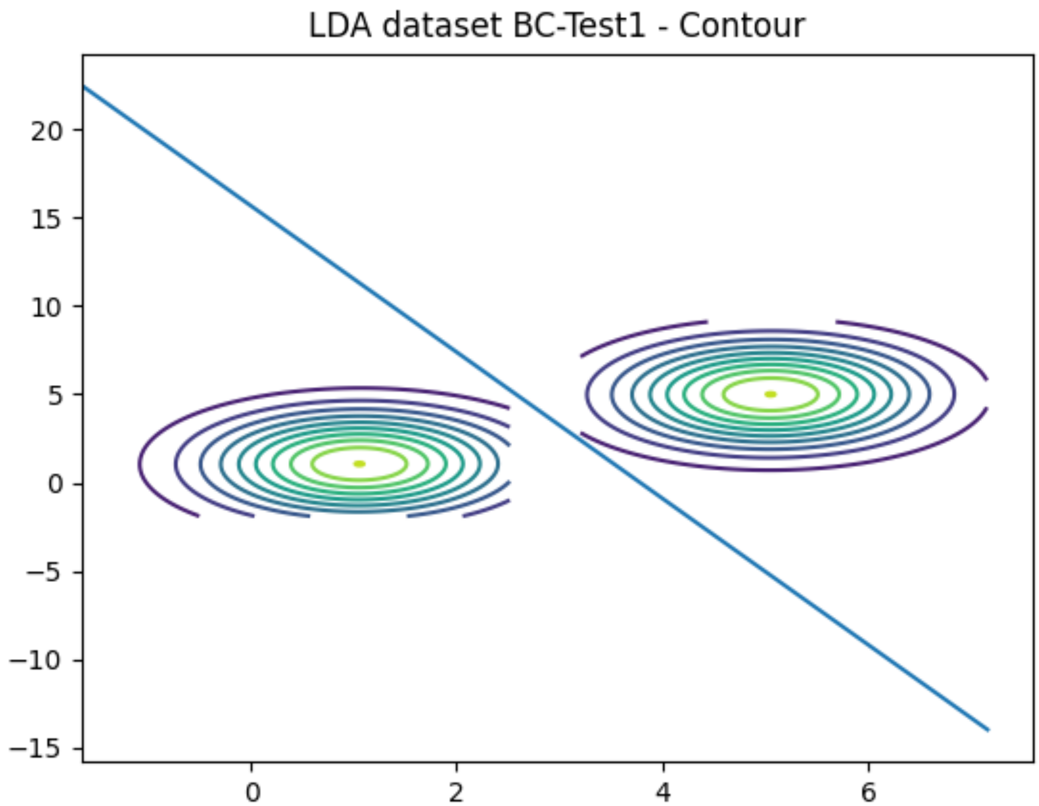
$\begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \begin{bmatrix} x_0 & x_1 \end{bmatrix} + b = 0$

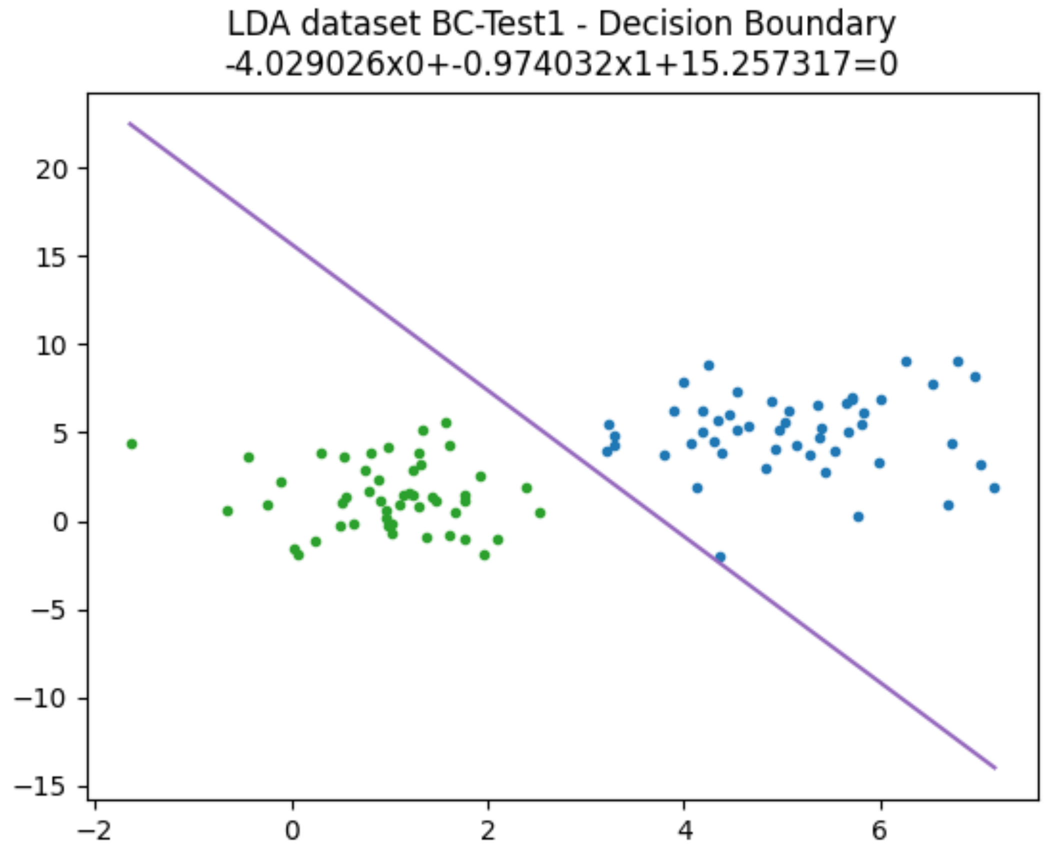
$a_0 x_0 + a_1 x_1 + b = 0$

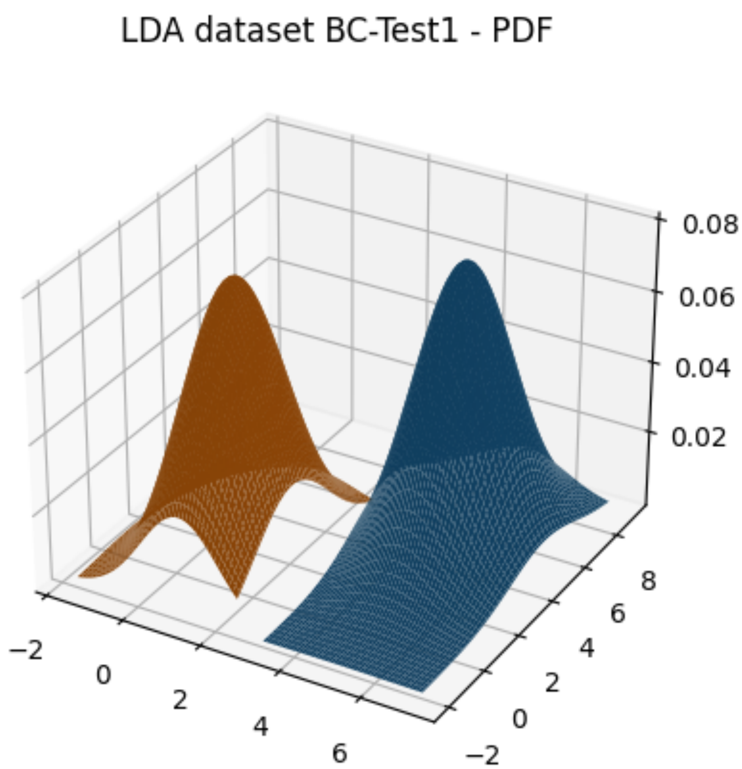
$x_1 = (-b - a_0 x_0)/a_1$

```
def plot_decision_boundary_logistic(X_input, classes_input, theta, title=None):
    min_x1_input = np.min(X_input[:, 1])
    max_x1_input = np.max(X_input[:, 1])
    class_0_input = np.array(classes_input[0])
    class_1_input = np.array(classes_input[1])
    plt.plot(class_0_input[:, 1], class_0_input[:, 2], '.', label='class 0')
    plt.plot(class_1_input[:, 1], class_1_input[:, 2], '.', label='class 1')
    plt.plot(
        [min_x1_input, max_x1_input], [
            -theta[1]/theta[2]*min_x1_input-theta[0]/theta[2],
            -theta[1]/theta[2]*max_x1_input-theta[0]/theta[2]],
        'r-', label='decision boundary')
    plt.title(title)
    plt.legend()
    plt.show()


def plot_decision_boundary_LDA(
        X_input, y_input, predicted_input, label_input,
        phi, mean, sigma, title=None):
    classes = []
    classes_corr = []
    classes_miss = []
    for label in label_input:
        where_label_i = np.where(y_input == label)[0]
        classes.append(X_input[where_label_i])
```

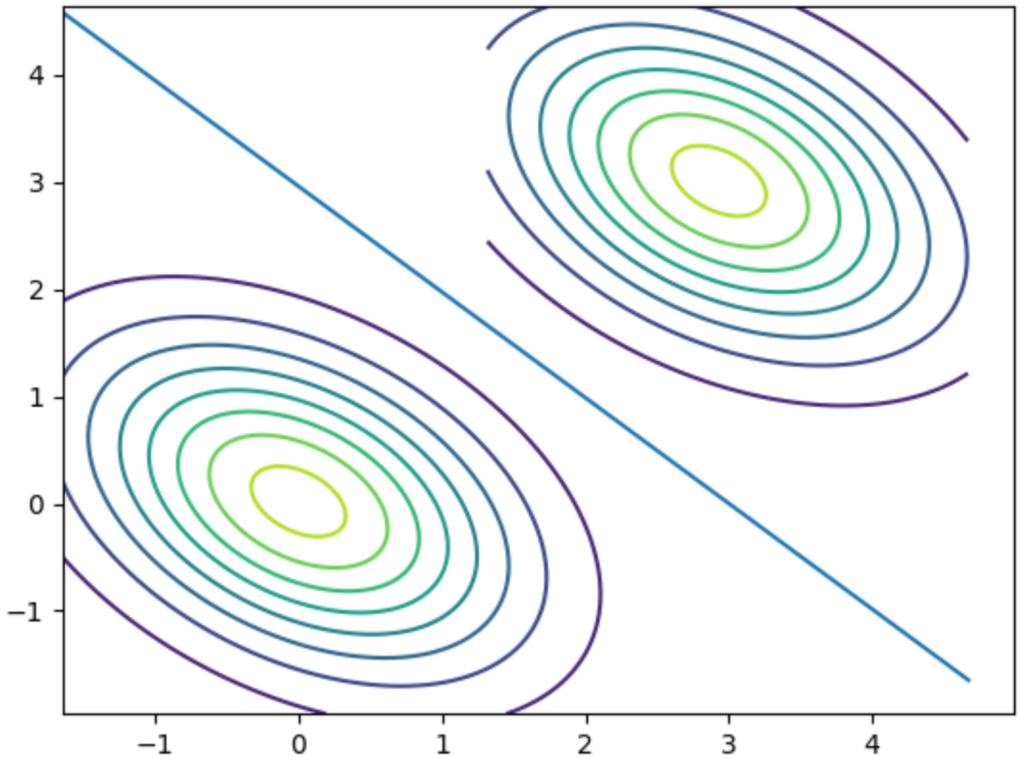
```
        where_label_i_corr = np.where(
            np.logical_and(predicted_input == y_input, y_input == label))[0]
        classes_corr.append(X_input[where_label_i_corr])
        where_label_i_miss = np.where(
            np.logical_and(predicted_input != y_input, y_input == label))[0]
        classes_miss.append(X_input[where_label_i_miss])
        plt.plot(classes_corr[label][:, 0], classes_corr[label][:, 1], '.')
        plt.plot(classes_miss[label][:, 0], classes_miss[label][:, 1], 'x')
    for label1 in label_input:
        for label2 in label_input[label_input < label1]:
            sigma_inverse = np.linalg.inv(sigma)
            b = (
                -0.5 * mean[label1] @ sigma_inverse @ mean[label1].T +
                0.5 * mean[label2] @ sigma_inverse @ mean[label2].T +
                np.log(phi[label1]/phi[label2]))
            a = np.linalg.inv(sigma) @ (mean[label1] - mean[label2]).T
            min_x0 = np.min(np.concatenate(
                (classes[label1][:, 0], classes[label2][:, 0])))
            max_x0 = np.max(np.concatenate(
                (classes[label1][:, 0], classes[label2][:, 0])))
            min_x1 = ((-b-a[0]*min_x0)/a[1])[0]
            max_x1 = ((-b-a[0]*max_x0)/a[1])[0]
            plt.plot([min_x0, max_x0], [min_x1, max_x1], '-')
    plt.title(
        title +
        f'\n{np.round(a[0][0], 6)}x0+{np.round(a[1][0], 6)}x1' +
        f'+{np.round(b[0][0], 6)}=0')
    plt.show()


def plot_pdf_LDA(
        X_input, y_input, label_input,
        phi, mean, sigma, title=None):
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    for label in label_input:
        class_label = X_input[(y_input == label).flatten()]
        x0_mesh, x1_mesh = np.mgrid[
            np.min(class_label[:, 0]):np.max(class_label[:, 0]):0.01,
            np.min(class_label[:, 1]):np.max(class_label[:, 1]):0.01]
        pos = np.dstack((x0_mesh, x1_mesh))
        prob = multivariate_normal(mean[label][0], sigma).pdf(pos)
        ax.plot_surface(x0_mesh, x1_mesh, prob)
    plt.title(title)
    plt.show()
```

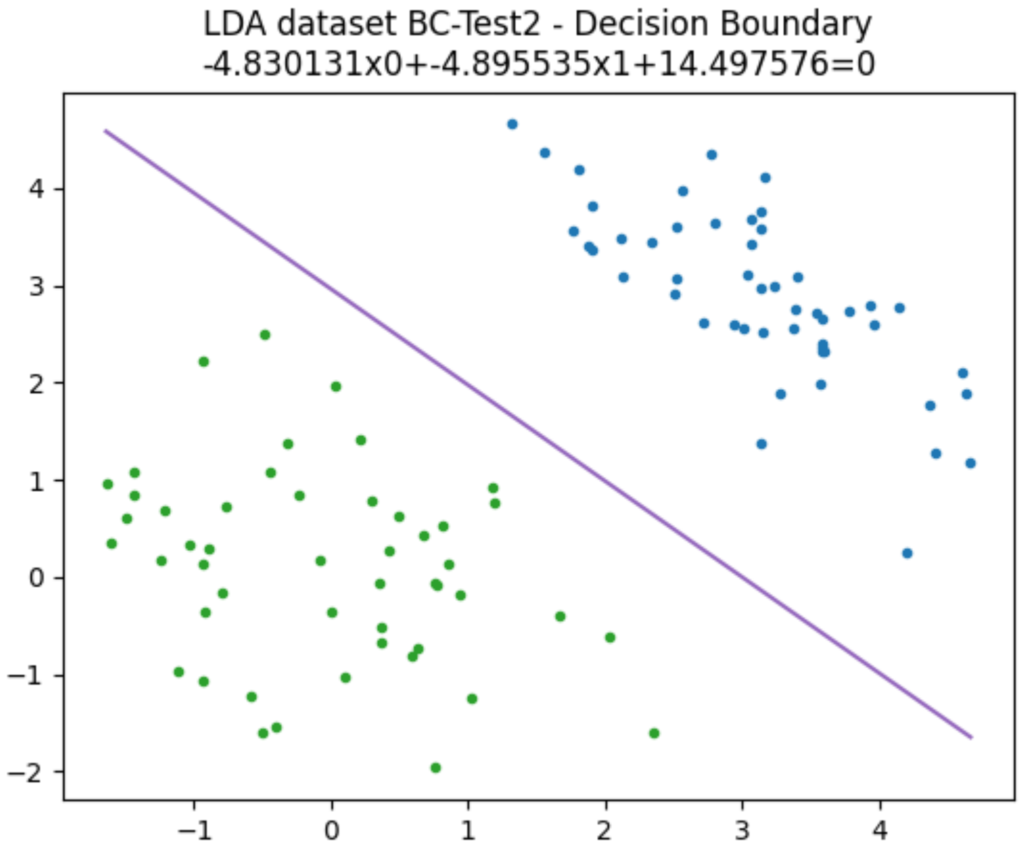

LDA dataset BC-Test1 - Contour
20
15
10
5
0
−5
−10
−15
0
2
4
6


LDA dataset BC-Test1 - Decision Boundary
-4.029026x0+-0.974032x1+15.257317=0
20
15
10
5
0
−5
−10
−15
−2
0
2
4
6


LDA dataset BC-Test1 - PDF
0.08
0.06
0.04
0.02
−2
0
2
4
6
−2
0
2
4
6
8

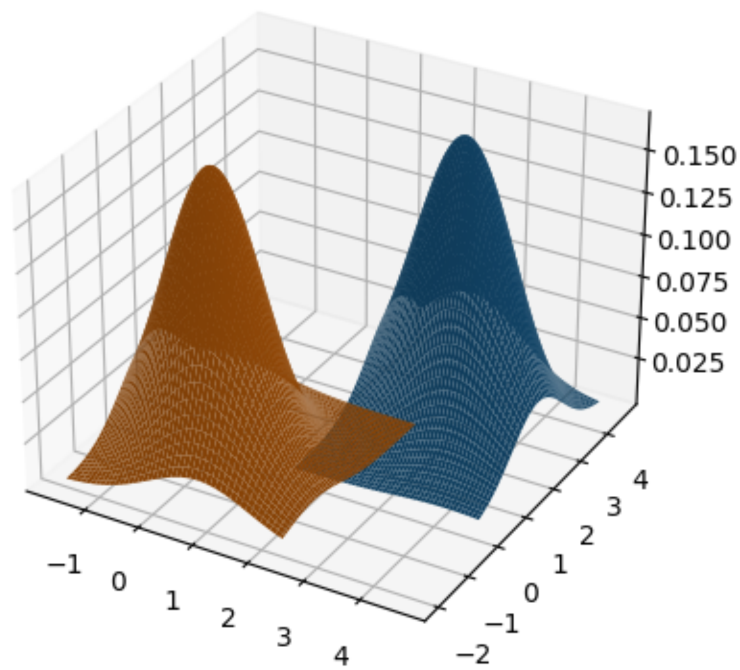
LDA dataset BC-Test2 - Contour



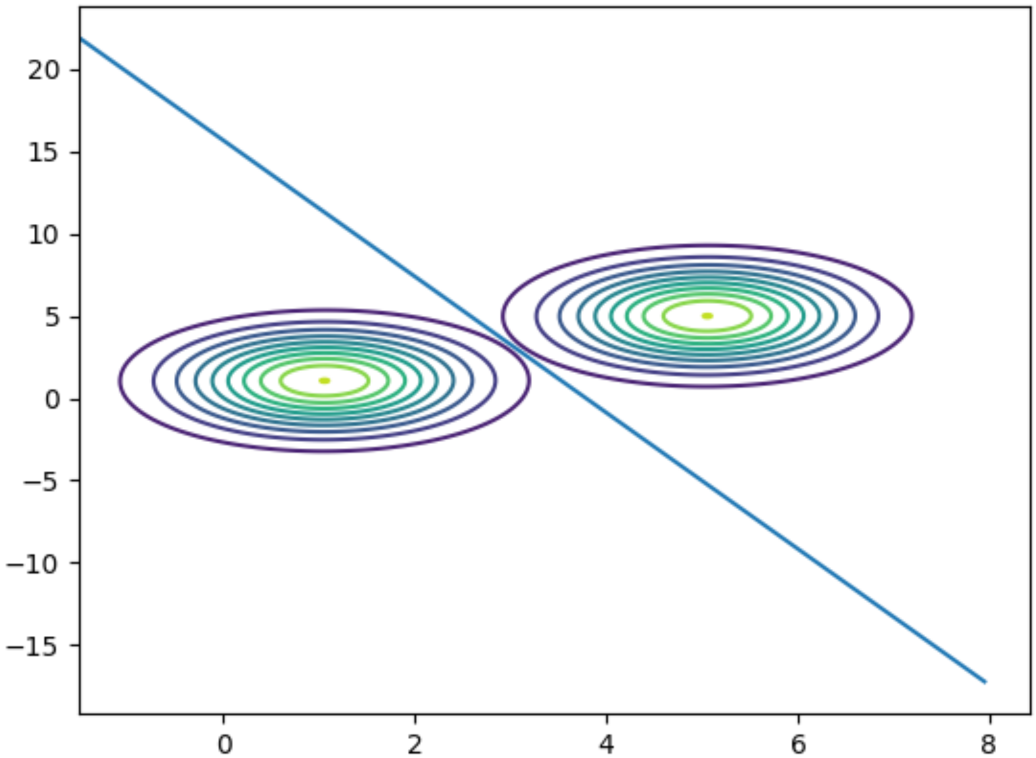


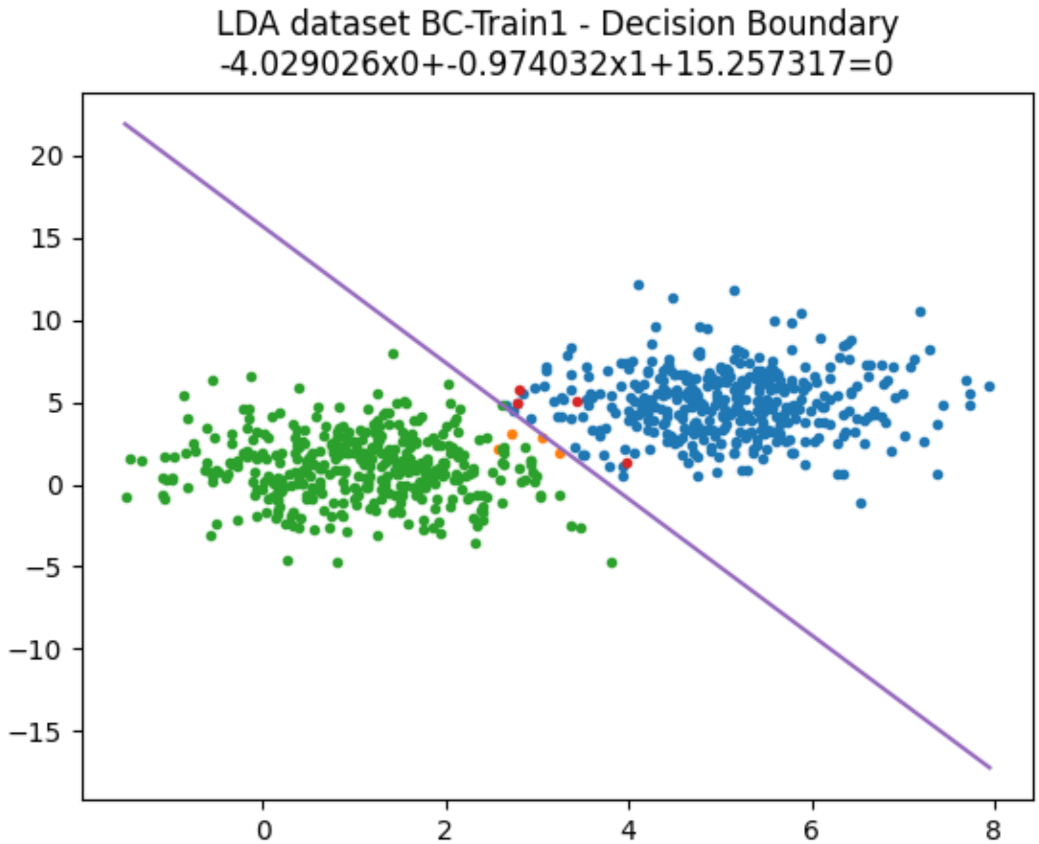
LDA dataset BC-Test2 - Decision Boundary
-4.830131x0+-4.895535x1+14.497576=0
4
3
2
1
0
−1
−2
−1
0
1
2
3
4

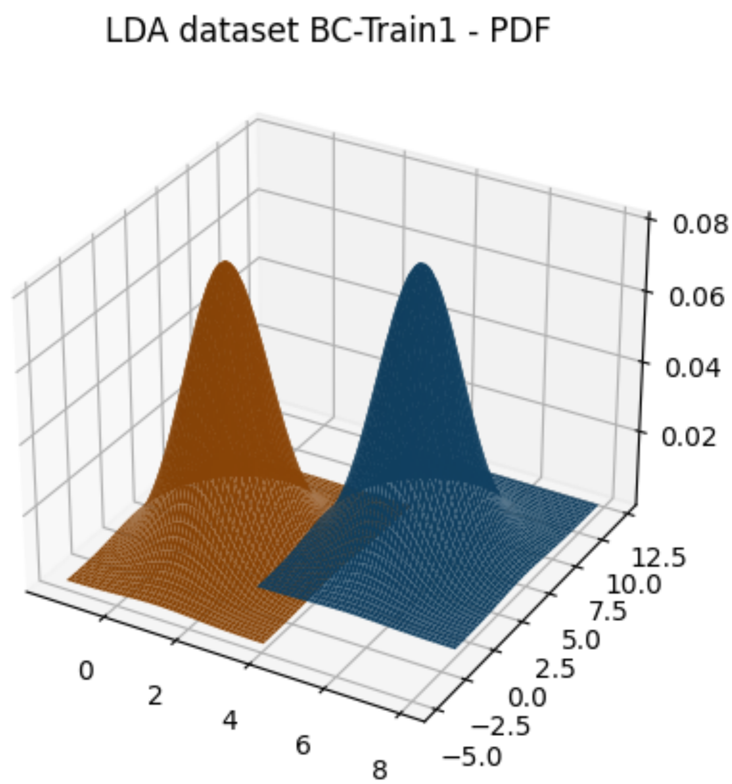
LDA dataset BC-Test2 - PDF

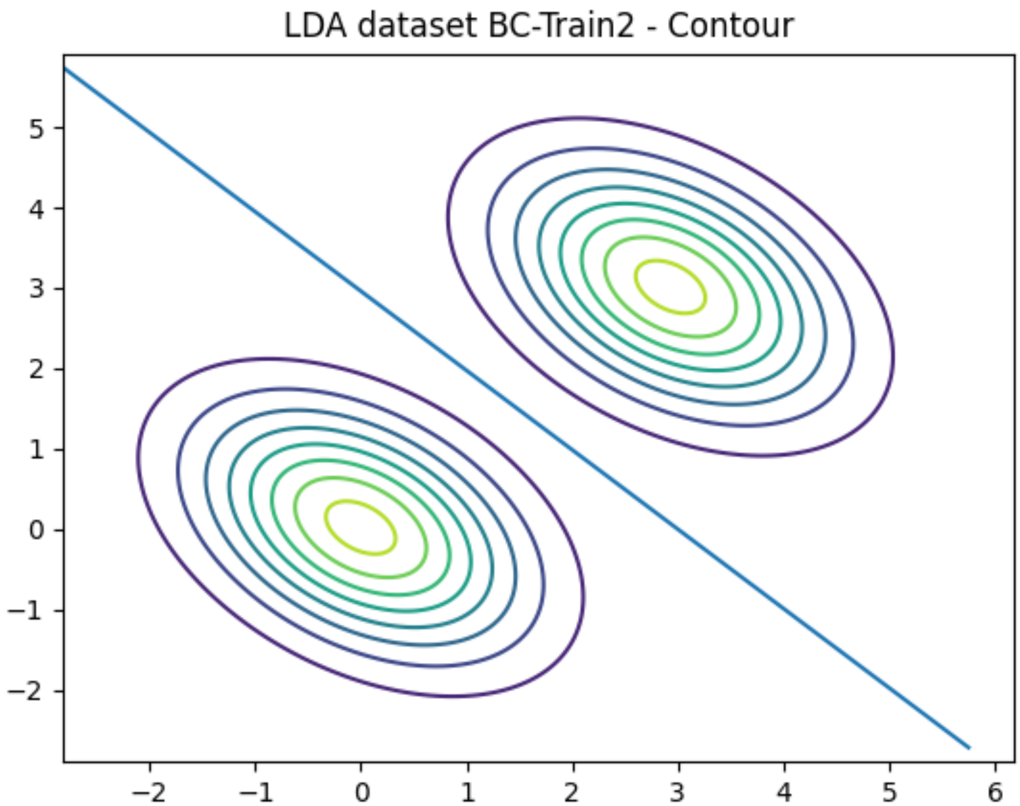


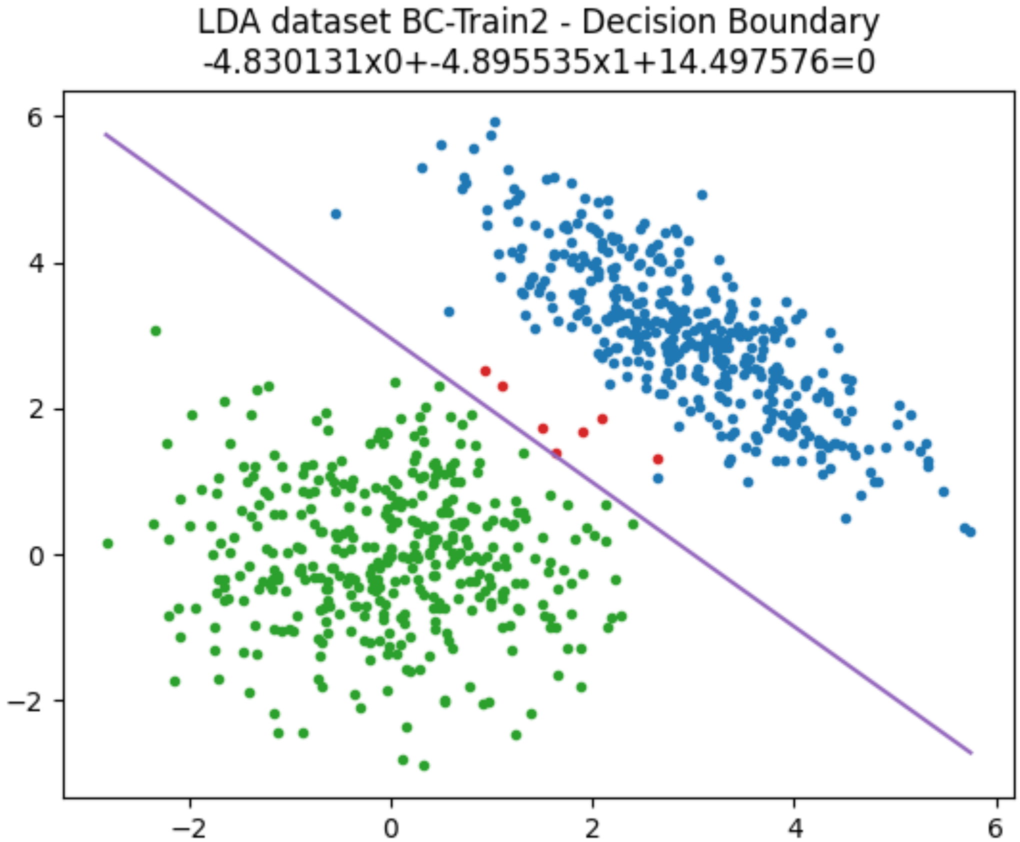
LDA dataset BC-Train1 - Contour



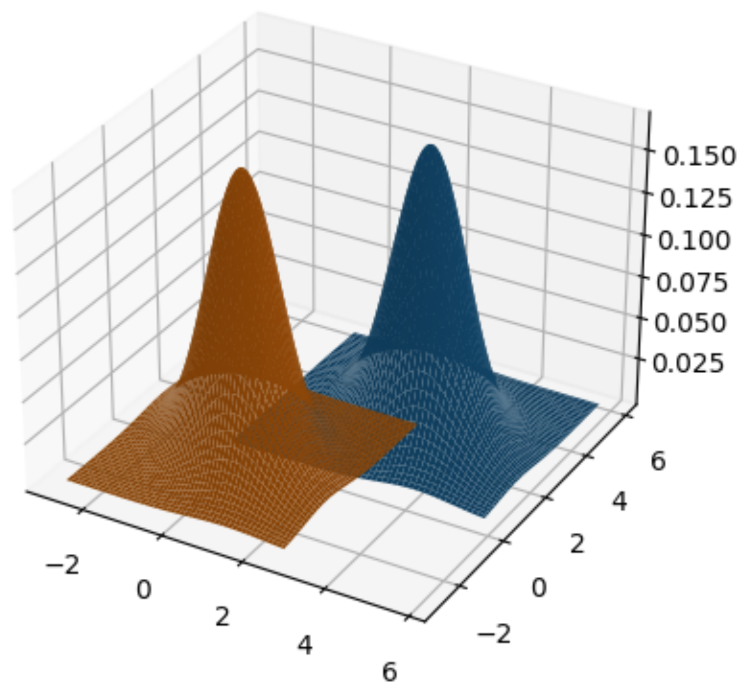

LDA dataset BC-Train1 - Decision Boundary
-4.029026x0+-0.974032x1+15.257317=0
20
15
10
5
0
−5
−10
−15
0
2
4
6
8


LDA dataset BC-Train1 - PDF
0.08
0.06
0.04
0.02
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
0
2
4
6
8


LDA dataset BC-Train2 - Contour
5
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4
5
6


LDA dataset BC-Train2 - Decision Boundary
-4.830131x0+-4.895535x1+14.497576=0
6
4
2
0
−2
−2
0
2
4
6

LDA dataset BC-Train2 - PDF



Quadratic Discriminant Analysis

$$P(\mathbf{X}|y=i) = \frac{1}{(\sqrt{2\pi})^n |\Sigma_i|^{\frac{1}{2}}} \; exp(\frac{-1}{2}(\mathbf{X}-\boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{X}-\boldsymbol{\mu}_i))$$

```
    def find_sigma_QDA(self):
        '''
        find sigma
        '''
        sigma = []
        for label in self.diffrent_label:
            where_label = np.where(self.y_train == label)[0]
            X_demean_class_i = (
                self.X_train[where_label] - self.mean_class[label])
            sigma.append(np.cov(X_demean_class_i.T))
        return sigma

    def find_parameters_QDA(self):
        self.phi = (
            self.count_diffrent_label_train / self.number_of_sample_train)
        self.mean_class = self.find_mean_classes()
        self.sigma = self.find_sigma_QDA()

    def find_probabilities_QDA(self, X_input):
        number_of_sample = X_input.shape[0]
        probabilities = np.zeros((number_of_sample, self.number_of_class))
        for label in self.diffrent_label:
            coefficient = 1 / (
                (2 * np.pi) ** (self.number_of_feature/2)
                * np.sqrt(abs(np.linalg.det(self.sigma[label]))))
            sigma_i_inverse = np.linalg.inv(self.sigma[label])
            X_demean = X_input - self.mean_class[label]
            P_X_given_y = coefficient * -0.5 * np.exp(
                    ((X_demean @ sigma_i_inverse) * X_demean)
                    @ np.ones((self.number_of_feature, 1)))
            prior = self.phi[label]
            probabilities[:, label:label+1] = P_X_given_y * prior
        return probabilities
```

In dataset my_QDA_dataset1 accuracy train is 0.7666666666666667 and accuracy test is 0.77
In dataset my_QDA_dataset2 accuracy train is 0.7666666666666667 and accuracy test is 0.77

$$log\frac{P(y=i)}{P(y=j)} - \frac{1}{2}log\frac{\left|\Sigma_i\right|}{\left|\Sigma_j\right|} - \frac{1}{2}[\mathbf{X}^T(\Sigma_i^{-1} - \Sigma_j^{-1})\mathbf{X} + \boldsymbol{\mu}_i^T\Sigma_i^{-1}\boldsymbol{\mu}_i$$
$$- \boldsymbol{\mu}_j^T\Sigma_j^{-1}\boldsymbol{\mu}_j - 2\mathbf{X}^T(\Sigma_i^{-1}\boldsymbol{\mu}_i - \Sigma_j^{-1}\boldsymbol{\mu}_j)] = 0$$

$$\Rightarrow \quad \mathbf{X}^T a\mathbf{X} + b^T\mathbf{X} + c = 0$$

In QDA:

$$x^T a x + b^T x + c = 0$$

$$\underbrace{[x_0 \;\; x_1]\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11}\end{bmatrix}}\begin{bmatrix} x_0 \\ x_1\end{bmatrix} + [b_0 \;\; b_1]\begin{bmatrix} x_0 \\ x_1\end{bmatrix} + c = 0$$

$$\underbrace{[x_0a_{00} + x_1a_{10},\; x_0a_{01} + x_1a_{11}]}$$

$$x_0^2a_{00} + x_0x_1a_{10} + x_0x_1a_{01} + x_1^2a_{11} + x_0b_0 + x_1b_1 + c = 0$$

$$\underbrace{x_1^2a_{11}}_{a} + \underbrace{x_1(x_0(a_{10} + a_{01}) + b_1)}_{b} + \underbrace{x_0^2a_{00} + x_0b_0 + c}_{c} = 0$$

$$ax^2 + bx + c = 0 \rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

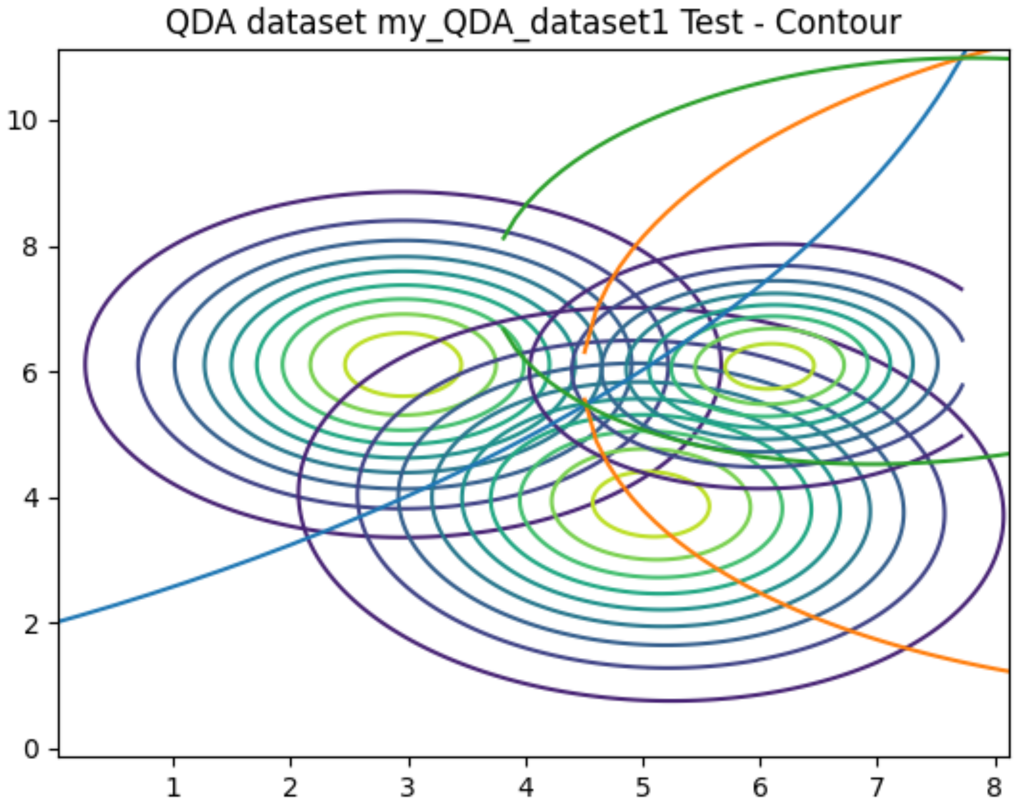
$$\mathbf{X}^T a\mathbf{X} + b^T\mathbf{X} + c = 0$$

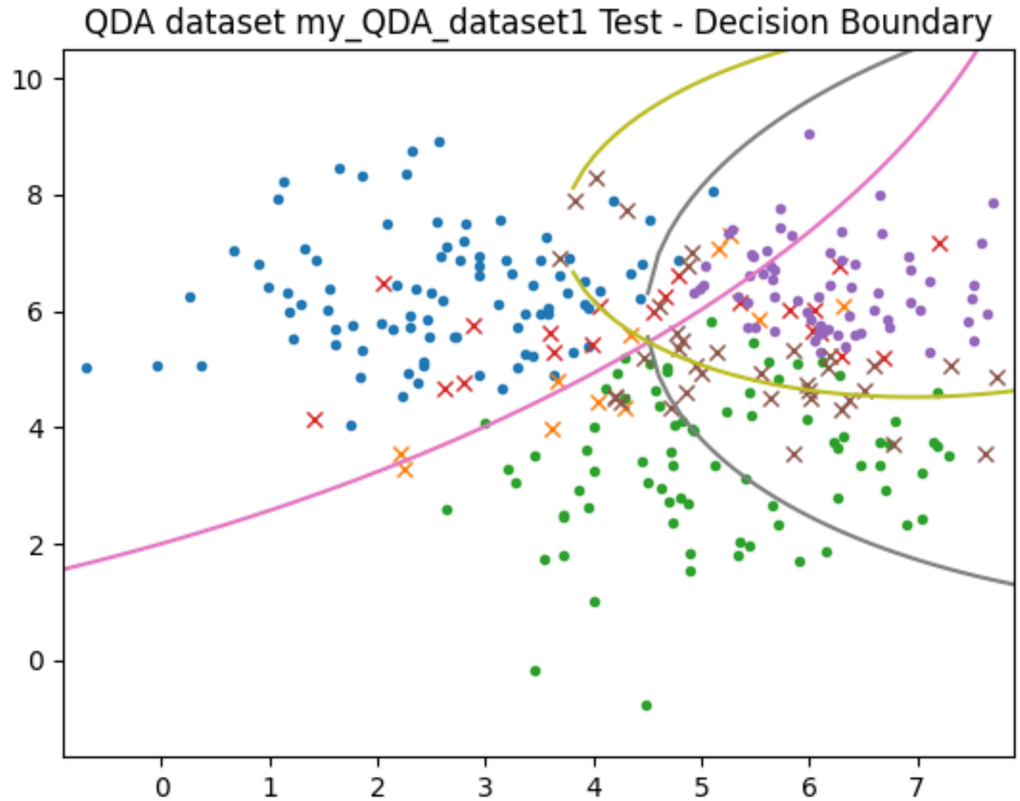
```python
def plot_contour_LDA(
        X_input, y_input, label_input,
        phi, mean, sigma, title=None):
    for label in label_input:
        class_label = X_input[(y_input == label).flatten()]
        x0_mesh, x1_mesh = np.mgrid[
            np.min(class_label[:, 0]):np.max(class_label[:, 0]):0.01,
            np.min(class_label[:, 1]):np.max(class_label[:, 1]):0.01]
        pos = np.dstack((x0_mesh, x1_mesh))
        plt.contour(
            x0_mesh, x1_mesh, multivariate_normal(
                mean[label][0], sigma).pdf(pos), levels=10)
    for label1 in label_input:
        for label2 in label_input[label_input < label1]:
            sigma_inverse = np.linalg.inv(sigma)
            b = (
                -0.5 * mean[label1] @ sigma_inverse @ mean[label1].T +
                0.5 * mean[label2] @ sigma_inverse @ mean[label2].T +
                np.log(phi[label1]/phi[label2]))
            a = np.linalg.inv(sigma) @ (mean[label1] - mean[label2]).T
            min_x0 = (np.min(np.concatenate(
                (X_input[(y_input == label1).flatten()][:, 0],
                 X_input[(y_input == label2).flatten()][:, 0]))))
            max_x0 = (np.max(np.concatenate(
                (X_input[(y_input == label1).flatten()][:, 0],
                 X_input[(y_input == label2).flatten()][:, 0]))))
            min_x1 = ((-b-a[0]*min_x0)/a[1])[0]
            max_x1 = ((-b-a[0]*max_x0)/a[1])[0]
            plt.plot([min_x0, max_x0], [min_x1, max_x1], '-')
    plt.title(title)
    plt.show()


def find_x1(a, b, c, x0):
    ap = a[1][1]
    bp = x0*(a[1][0]+a[0][1])+b[1]
    cp = (x0**2)*a[0][0]+x0*b[0]+c
    # ap*X^2 + bp*X + c = 0
    if(bp**2-4*ap*cp >= 0):
        return [((-bp-np.sqrt(bp**2-4*ap*cp))/(2*ap))[0][0],
                ((-bp+np.sqrt(bp**2-4*ap*cp))/(2*ap))[0][0]]
    return None


def plot_decision_boundary_QDA(
        X_input, y_input, predicted_input, label_input,
        phi, mean, sigma, title=None):
    classes = []
```

```python
    classes_corr = []
    classes_miss = []
    for label in label_input:
        where_label_i = np.where(y_input == label)[0]
        classes.append(X_input[where_label_i])
        where_label_i_corr = np.where(
            np.logical_and(predicted_input == y_input, y_input == label))[0]
        classes_corr.append(X_input[where_label_i_corr])
        where_label_i_miss = np.where(
            np.logical_and(predicted_input != y_input, y_input == label))[0]
        classes_miss.append(X_input[where_label_i_miss])
        plt.plot(classes_corr[label][:, 0], classes_corr[label][:, 1], '.')
        plt.plot(classes_miss[label][:, 0], classes_miss[label][:, 1], 'x')
    for label1 in label_input:
        for label2 in label_input[label_input < label1]:
            sigma1_inverse = np.linalg.inv(sigma[label1])
            sigma2_inverse = np.linalg.inv(sigma[label2])
            det1 = np.linalg.det(sigma[label1])
            det2 = np.linalg.det(sigma[label2])
            a = -0.5*(sigma1_inverse-sigma2_inverse)
            b = (
                (sigma1_inverse@mean[label1].T) -
                (sigma2_inverse@mean[label2].T))
            c = (
                np.log(phi[label1]/phi[label2])-0.5*(np.log(det1/det2)) +
                (-0.5*(mean[label1]@sigma1_inverse)@mean[label1].T) +
                (0.5*(mean[label2]@sigma2_inverse)@mean[label2].T))
            min_x0 = np.min(np.concatenate(
                (classes[label1][:, 0], classes[label2][:, 0])))
            max_x0 = np.max(np.concatenate(
                (classes[label1][:, 0], classes[label2][:, 0])))
            x0 = []
            x1_u = []
            x1_d = []
            for x in np.arange(min_x0-0.5, max_x0+0.5, 0.1):
                tmp = find_x1(a, b, c, x)
                if tmp is not None:
                    x0.append(x)
                    x1_u.append(tmp[0])
                    x1_d.append(tmp[1])
            x0_nw = np.concatenate((x0, x0[::-1]))
            x1_nw = np.concatenate((x1_d, x1_u[::-1]))
            plt.plot(x0_nw, x1_nw, '-')
    plt.title(title)
    plt.show()


def plot_pdf_QDA(
```

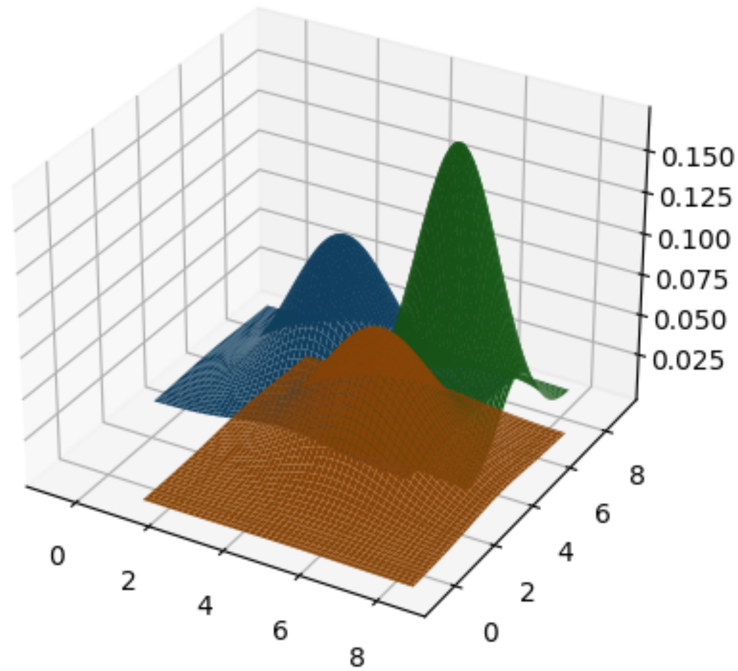
```python
        X_input, y_input, label_input,
        phi, mean, sigma, title=None):
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    for label in label_input:
        class_label = X_input[(y_input == label).flatten()]
        x0_mesh, x1_mesh = np.mgrid[
            np.min(class_label[:, 0]):np.max(class_label[:, 0]):0.01,
            np.min(class_label[:, 1]):np.max(class_label[:, 1]):0.01]
        pos = np.dstack((x0_mesh, x1_mesh))
        prob = multivariate_normal(mean[label][0], sigma[label]).pdf(pos)
        ax.plot_surface(x0_mesh, x1_mesh, prob)
    plt.title(title)
    plt.show()


def plot_contour_QDA(
        X_input, y_input, label_input,
        phi, mean, sigma, title=None):
    for label in label_input:
        class_label = X_input[(y_input == label).flatten()]
        x0_mesh, x1_mesh = np.mgrid[
            np.min(class_label[:, 0]):np.max(class_label[:, 0]):0.01,
            np.min(class_label[:, 1]):np.max(class_label[:, 1]):0.01]
        pos = np.dstack((x0_mesh, x1_mesh))
        plt.contour(
            x0_mesh, x1_mesh, multivariate_normal(
                mean[label][0], sigma[label]).pdf(pos), levels=10)
    for label1 in label_input:
        for label2 in label_input[label_input < label1]:
            sigma1_inverse = np.linalg.inv(sigma[label1])
            sigma2_inverse = np.linalg.inv(sigma[label2])
            det1 = np.linalg.det(sigma[label1])
            det2 = np.linalg.det(sigma[label2])
            a = -0.5*(sigma1_inverse-sigma2_inverse)
            b = (
                (sigma1_inverse@mean[label1].T) -
                (sigma2_inverse@mean[label2].T))
            c = (
                np.log(phi[label1]/phi[label2])-0.5*(np.log(det1/det2)) +
                (-0.5*(mean[label1]@sigma1_inverse)@mean[label1].T) +
                (0.5*(mean[label2]@sigma2_inverse)@mean[label2].T))
            min_x0 = (np.min(np.concatenate(
                (X_input[(y_input == label1).flatten()][:, 0],
                 X_input[(y_input == label2).flatten()][:, 0]))))
            max_x0 = (np.max(np.concatenate(
                (X_input[(y_input == label1).flatten()][:, 0],
                 X_input[(y_input == label2).flatten()][:, 0]))))
```

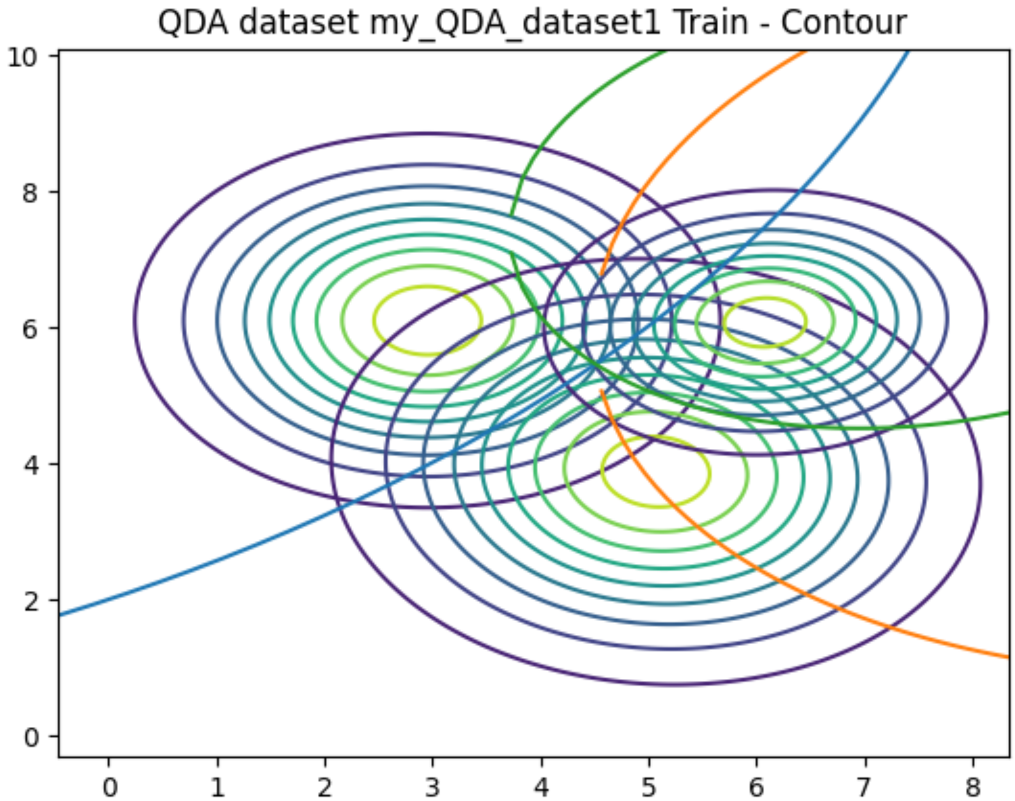
```
        x0 = []
        x1_u = []
        x1_d = []
        for x in np.arange(min_x0-0.5, max_x0+0.5, 0.1):
            tmp = find_x1(a, b, c, x)
            if tmp is not None:
                x0.append(x)
                x1_u.append(tmp[0])
                x1_d.append(tmp[1])
        x0_nw = np.concatenate((x0, x0[::-1]))
        x1_nw = np.concatenate((x1_d, x1_u[::-1]))
        plt.plot(x0_nw, x1_nw, '-')
```

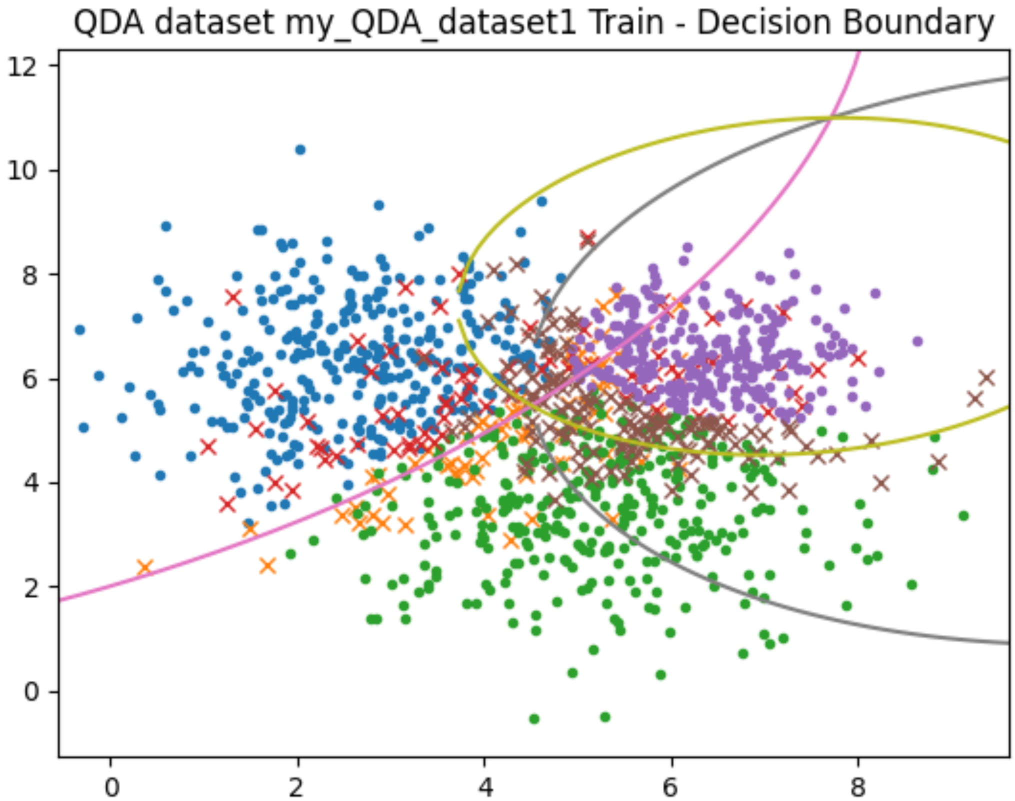

QDA dataset my_QDA_dataset1 Test - Contour
0
2
4
6
8
10
1
2
3
4
5
6
7
8


QDA dataset my_QDA_dataset1 Test - Decision Boundary
10
8
6
4
2
0
0
1
2
3
4
5
6
7

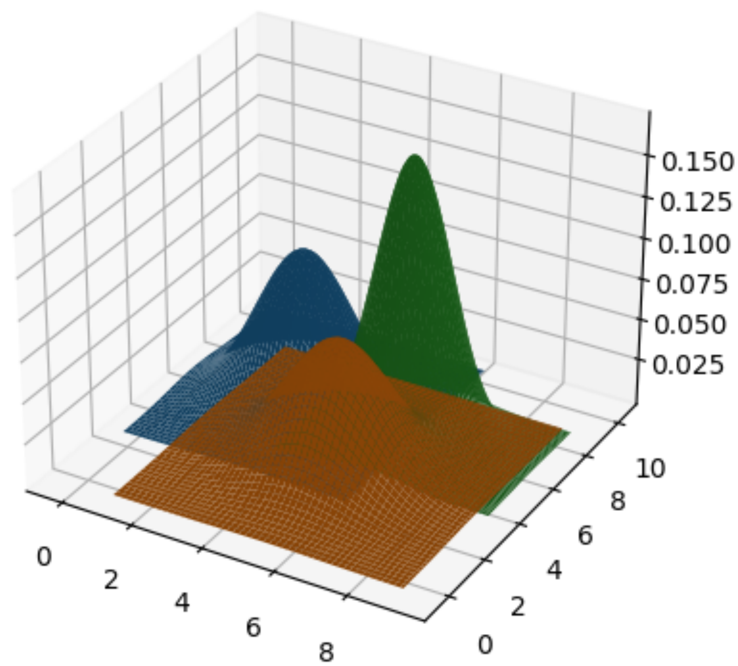
QDA dataset my_QDA_dataset1 Test - pdf

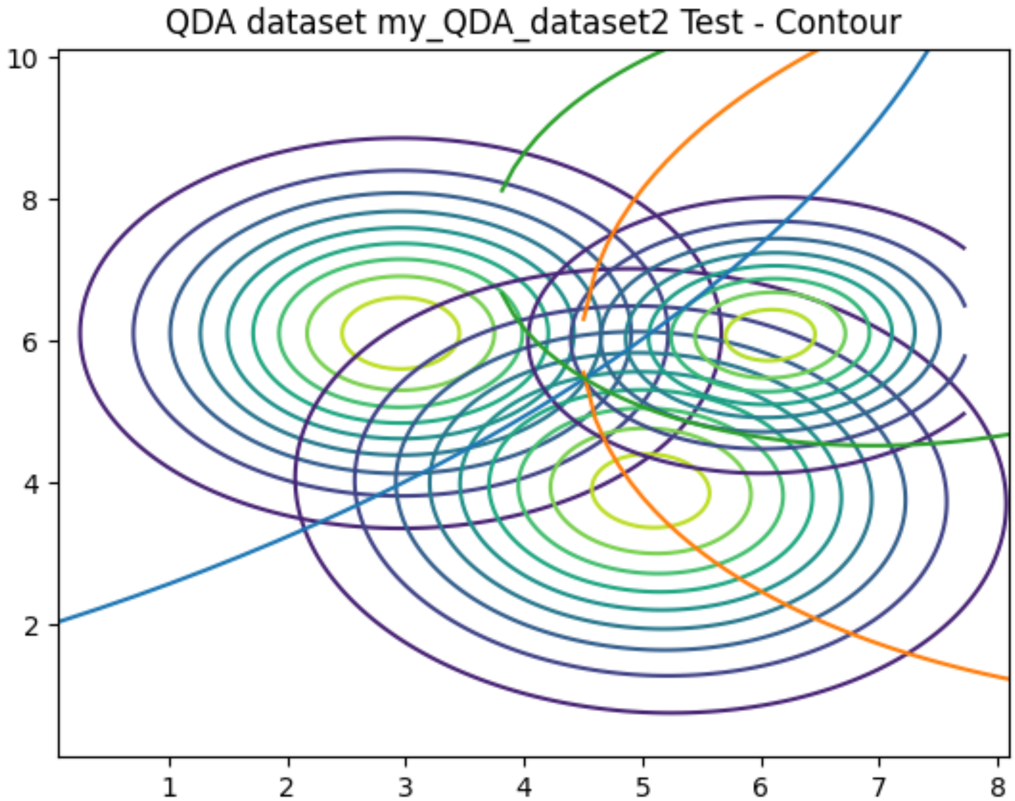


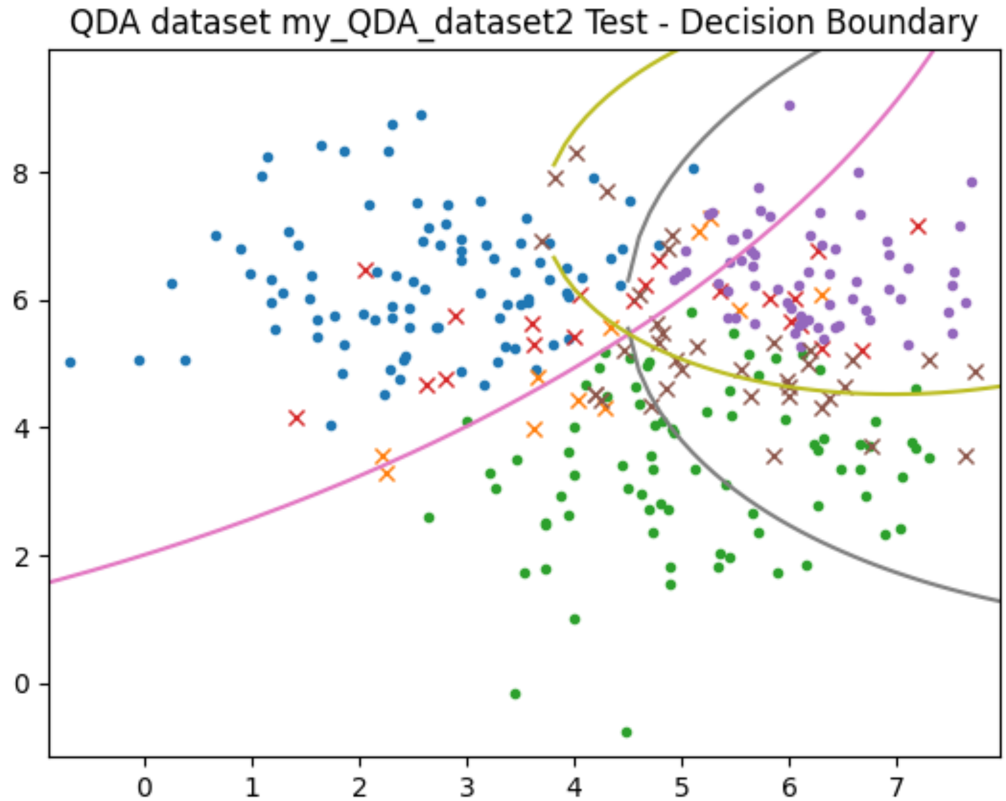

QDA dataset my_QDA_dataset1 Train - Contour
10
8
6
4
2
0
0
1
2
3
4
5
6
7
8


QDA dataset my_QDA_dataset1 Train - Decision Boundary
12
10
8
6
4
2
0
0
2
4
6
8

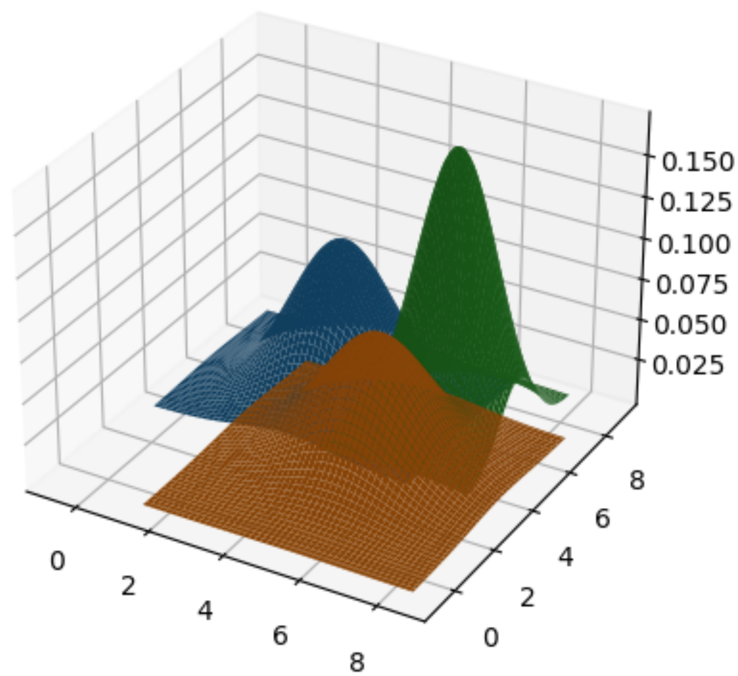
QDA dataset my_QDA_dataset1 Train - pdf

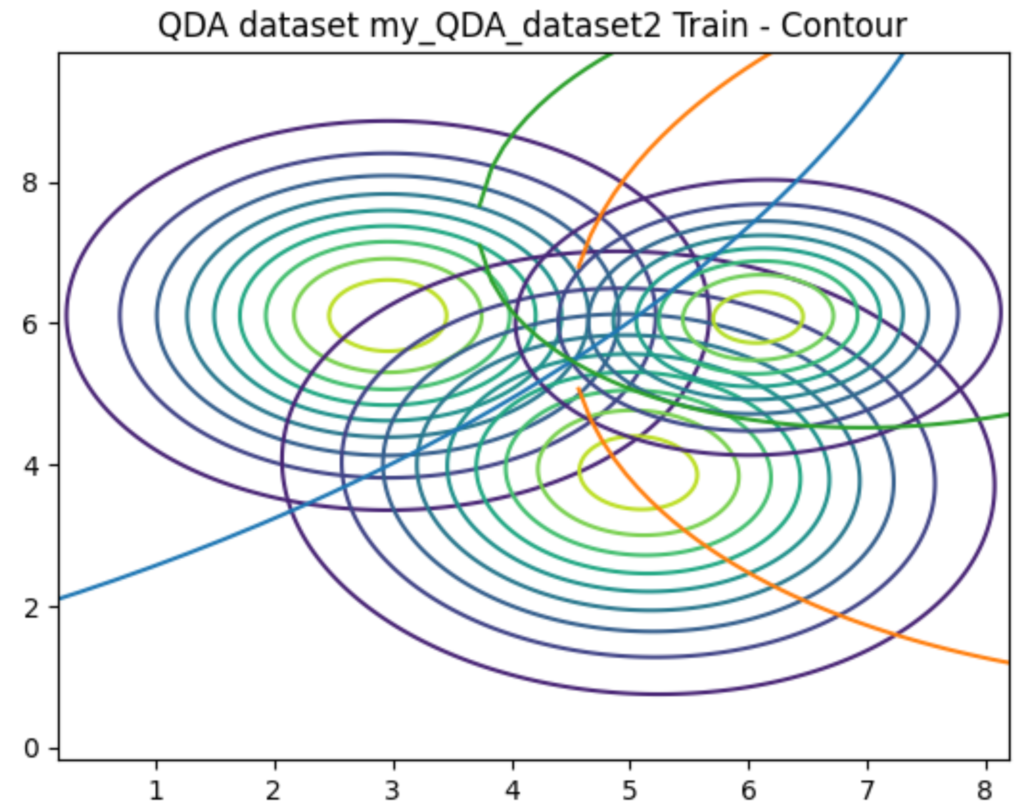


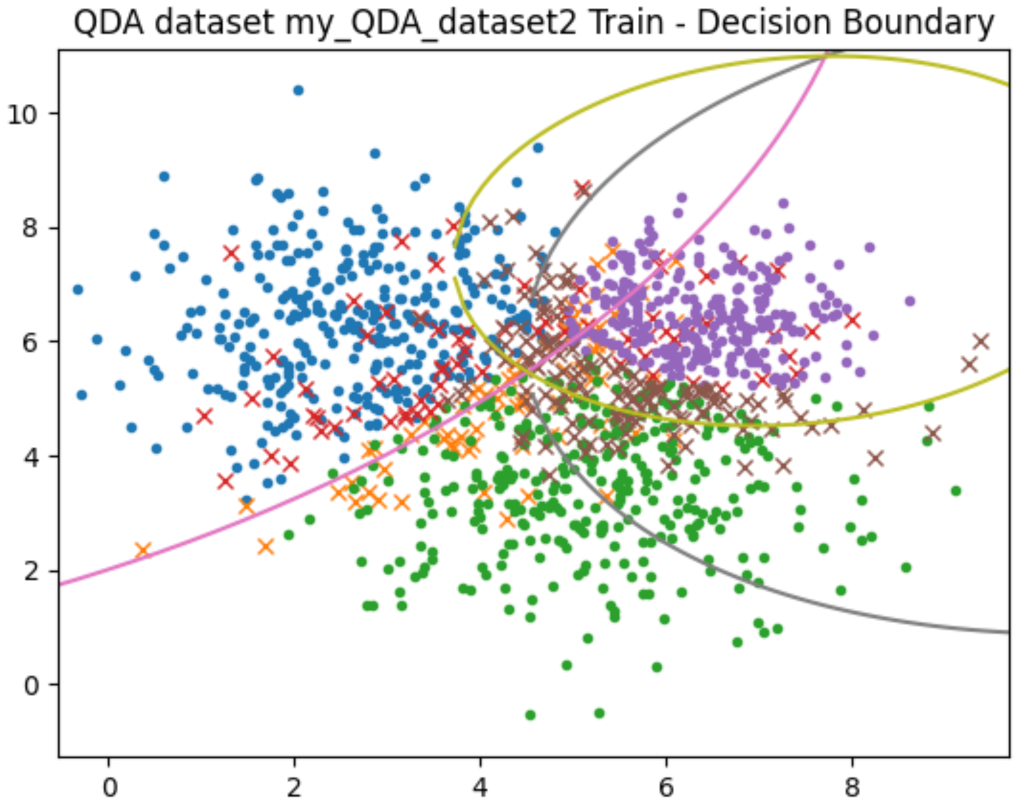

QDA dataset my_QDA_dataset2 Test - Contour
10
8
6
4
2
1
2
3
4
5
6
7
8


QDA dataset my_QDA_dataset2 Test - Decision Boundary
0
2
4
6
8
0
1
2
3
4
5
6
7

QDA dataset my_QDA_dataset2 Test - pdf






QDA dataset my_QDA_dataset2 Train - Decision Boundary
10
8
6
4
2
0
0
2
4
6
8

QDA dataset my_QDA_dataset2 Train - pdf