Problem : 2 :

i) We know that the likelihood function is as follows:

$$L(w) = \prod_{n=1}^{N} p(c_1|x_n)^{y_n} (1 - p(c_1|x_n))^{1-y_n}$$

Taking the negative logarithm,

$$-\ln(L(w)) = -\ln\left(\prod_{n=1}^{N} p(c_1|x_n)^{y_n} (1 - p(c_1|x_n))^{1-y_n}\right)$$

$$= -\sum_{n=1}^{N} \left(y_n \ln(p(c_1|x_n)) + (1-y_n) \ln(1-p(c_1|x_n))\right) \quad \text{— (1)}$$

Now, we also know that,

$$p(c_1|x) = \sigma(w^T x + w_0) = f(x)$$

$$\therefore p(c_1|x_n) = \sigma(w^T x_n + w_0) = f(x_n) \quad \text{— (2)}$$

We know that,

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

Suppose $a = w^T x + w_0$.

$$\therefore \sigma(a) = \frac{1}{1 + e^{-a}} = \frac{1}{1 + e^{-(w^T + w_0)}}$$

Derivating the sigmoid function with respect to w.

$$\sigma(a) = \frac{1}{1+e^{-a}}$$

$$\frac{\partial(\sigma(a))}{\partial w} = \frac{\partial}{\partial w}\left(\frac{1}{1+e^{-a}}\right)$$

$$= \frac{\partial}{\partial a}\left(\frac{1}{1+e^{-a}}\right) * \frac{\partial a}{\partial w} \quad \text{(property of derivatives)}$$

$$= \frac{1}{(1+e^{-a})^2} \cdot e^{-a} \cdot \frac{\partial a}{\partial w}$$

$$= \frac{e^{-a}}{(1+e^{-a})^2} \frac{\partial a}{\partial w}$$

$$= \frac{1}{(1+e^{-a})} \cdot \frac{e^{-a}}{(1+e^{-a})} \cdot \frac{\partial a}{\partial w}$$

$$= \frac{1}{(1+e^{-a})} \cdot \left(1 - \frac{1}{(1+e^{-a})}\right) \cdot \frac{\partial a}{\partial w}$$

$$= \sigma(a)(1-\sigma(a)) \cdot \frac{\partial a}{\partial w}$$

$$\frac{\partial(\sigma(a))}{\partial w} = \sigma(a)(1-\sigma(a)) \cdot \frac{\partial a}{\partial w} \quad \text{---} \quad (3)$$

Now,

$$-\ln L(w) = E(w)$$

$$E(w) = -\sum_{n=1}^{N} \left(y_n \ln(\sigma(a)) + (1-y_n) \ln(1-\sigma(a)) \right)$$

Above equation can be derived from (2) and here we suppose that $a = w^T x_n + w_0$

$$E(w) = -\sum_{n=1}^{N} \left(y_n \ln(\sigma(a)) + (1-y_n) \ln(1-\sigma(a)) \right)$$

Taking derivative with respect to w,

$$\frac{\partial E(w)}{\partial w} = -\sum_{n=1}^{N} \left(y_n \frac{\partial}{\partial w} \ln(\sigma(a)) + (1-y_n) \frac{\partial \ln(1-\sigma(a))}{\partial w} \right)$$

$$= -\sum_{n=1}^{N} \left(y_n \frac{1}{\sigma(a)} \frac{\partial(\sigma(a))}{\partial w} + (1-y_n) \frac{1}{(1-\sigma(a))} \frac{\partial(1-\sigma(a))}{\partial w} \right)$$

$$= -\sum_{n=1}^{N} \left(y_n \frac{1}{\sigma(a)} \frac{\partial(\sigma(a))}{\partial w} + (1-y_n) \left(\frac{1}{\sigma(a)-1} \right) \frac{\partial(\sigma(a))}{\partial w} \right)$$

Substituting the values of (3) in the above equation.

$$= -\sum_{n=1}^{N} \left(y_n \frac{1}{\sigma(a)} \sigma(a)(1-\sigma(a)) \frac{\partial a}{\partial w} + \frac{(1-y_n)}{(\sigma(a)-1)} \sigma(a)(1-\sigma(a)) \frac{\partial a}{\partial w} \right)$$

(-1)

P.T.O.

Now we have assumed that,

$$a = w^T x_n + w_0$$

$$\frac{\partial a}{\partial w} = x_n \quad (14)$$

Now substituting value of $\frac{\partial a}{\partial w}$ in the derived equation,

$$= -\sum_{n=1}^{N} \left(y_n (1 - \sigma(a)) x_n + (y_n - 1) \sigma(a) \cdot x_n \right)$$

$$= -\sum_{n=1}^{N} \left(y_n (1 - f(x_n)) x_n + (y_n - 1)(f(x_n) \cdot x_n) \right)$$

Because $p(C_1 | x_n) = \sigma(a) = f(x_n)$,

$$= -\sum_{n=1}^{N} y_n x_n - y_n f(x_n) \cdot x_n + y_n f(x_n) \cdot x_n - f(x_n) \cdot x_n$$

$$= -\sum_{n=1}^{N} x_n (y_n - f(x_n))$$

$$= \sum_{n=1}^{N} (f(x_n) - y_n) \cdot x_n$$

$$\therefore \nabla_w E(w) = \sum_{n=1}^{N} (f(x_n) - y_n) x_n$$

Problem : 3

2)

a) finding first principal component

(-2, -2) , (0, 0) , (2, 2) are given points.

$$\bar{x} = \frac{\sum x}{n} = \frac{-2+0+2}{3} = 0 \qquad \bar{x} = 0 \quad — (1)$$

$$\bar{y} = \frac{\sum y}{n} = \frac{-2+0+2}{3} = 0 \qquad \bar{y} = 0 \quad — (2)$$

using unbiased estimation of covariance,

$$var(x) = \frac{1}{N-1} \sum_{n=1}^{N} (x_n - \bar{x})^2$$

$$cov(x, y) = \frac{1}{N-1} \sum_{n=1}^{N} (x_n - \bar{x})(y_n - \bar{y})$$

$$cov(x, x) = var(x) = \frac{1}{2}((-2)^2 + (2)^2) = 4$$

$$cov(x, y) = \frac{1}{2}((-2)(-2) + (2)(2)) = 4$$

$$cov(y, y) = var(y) = \frac{1}{2}((-2)^2 + (2)^2) = 4$$

$$cov(y, x) = \frac{1}{2}((-2)(-2) + (2)(2)) = 4$$

We know that the covariance matrix is defined as follows:

$$M = \begin{bmatrix} cov(x,x) & cov(x,y) \\ cov(y,x) & cov(y,y) \end{bmatrix}$$

$$M = \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix}$$

Now for the eigenvalues, we know that

$$\det(M - \lambda I) = 0$$

$$\det\left(\begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}\right) = 0$$

$$\det\left(\begin{bmatrix} 4-\lambda & 4 \\ 4 & 4-\lambda \end{bmatrix}\right) = 0$$

$$\therefore (4-\lambda)^2 - (4)^2 = 0$$

$$\lambda^2 - 8\lambda + 16 - 16 = 0$$

$$\therefore \lambda^2 - 8\lambda = 0$$

$$\therefore \lambda(\lambda - 8) = 0$$

$\lambda_1 = 0$ and $\lambda_2 = 8$

Since $\lambda_2 > \lambda_1$, first component is given by $\lambda_2 = 8$

→ Eigen vector of the covariance matrix :

for $\lambda_1 = 0$,

$$\begin{bmatrix} 4-\lambda_1 & 4 \\ 4 & 4-\lambda_1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$$

$$\begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$$

$$\begin{bmatrix} 4x + 4y \\ 4x + 4y \end{bmatrix} = 0$$

$\therefore 4x + 4y = 0$

$x + y = 0$

This gives, $(x=1, y=-1)$, $(x=-1, y=1)$ and so on.

for $\lambda_2 = 8$

$$\begin{bmatrix} 4-\lambda_2 & 4 \\ 4 & 4-\lambda_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$$

$$\begin{bmatrix} -4 & 4 \\ 4 & -4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$$

$$\begin{bmatrix} -4x + 4y \\ 4x - 4y \end{bmatrix} = 0$$

$\therefore -4x + 4y = 0 \quad \therefore \quad x = y$

Hence this gives $(x=1, y=1)$, $(x=2, y=2)$ and so on

$\therefore$ eigen vector for λ_2 = $V_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Since this gives us the first component, we can normalize it,

$$|V_2| = \sqrt{(1)^2 + (1)^2}$$
$$= \sqrt{2}$$

$$\therefore e_2 = \frac{V_2}{|V_2|} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

Hence first principal component, $v = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}^T$ and it can also be written as $v = \begin{bmatrix} \frac{-1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix}^T$

b) projections of given points into the achieved 1D subspace and finding new coordinates.

Let P_1, P_2, P_3 be the new coordinates.

We know that,

$$P_1 = V^T \begin{bmatrix} x_1 - \bar{x} \\ y_1 - \bar{y} \end{bmatrix}$$

$$= \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$= \sqrt{2} + \sqrt{2} = 2\sqrt{2}$$

$$P_2 = V^T \begin{bmatrix} x_2 - \bar{x} \\ y_2 - \bar{y} \end{bmatrix}$$

$$= \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$= 0$$

$$P_3 = V^T \begin{bmatrix} x_3 - \bar{x} \\ y_3 - \bar{y} \end{bmatrix}$$

$$= \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} -2 \\ -2 \end{bmatrix}$$

$$= -\sqrt{2} - \sqrt{2} = -2\sqrt{2}$$

$p_1 = 2\sqrt{2}$ $\quad p_2 = 0$ $\quad p_3 = -2\sqrt{2}$

variance is defined as,

$$\text{var}(p) = \frac{1}{N-1} \sum_{i=1}^{N} (p_i - \bar{p})^2$$

$$\bar{p} = \frac{2\sqrt{2} - 2\sqrt{2} + 0}{3} = 0$$

$$\therefore \text{var}(p) = \frac{1}{2} \left((2\sqrt{2})^2 + (-2\sqrt{2})^2 \right)$$

$$= \frac{1}{2} (16) = 8$$

$\therefore$ variance of the new points is 8.

c) Cumulative explained variance can be defined as total variance of the principal component. Here we can say that the cumulative explained variance is 8. which we have already calculated in part b. Now to check what percentage of variance is being covered by the principal component we can just take the eigenvalue of principal component and divide it by all the eigen values. Here in our case we have $\lambda_1 = 0$ and $\lambda_2 = 8$ Hence we can say that principal component captures 100 % of the variance and there is no variance that is not captured by the principal component.

Problem : 4

i) Equation of SVM hyperplane here:

→ We know that points having $\alpha_i = 0$ are not support vectors.

→ Thus only the non-zero α_i determine the support vectors. ie x_1, x_4, x_7, x_9

weight vector $w = \sum_{\alpha_i > 0} \alpha_i y_i x_i$

$$= (0.414)\begin{pmatrix}4\\2.7\end{pmatrix} - (0.018)\begin{pmatrix}2.5\\1\end{pmatrix} + (0.018)\begin{pmatrix}3.5\\4\end{pmatrix} - (0.414)\begin{pmatrix}2\\2.1\end{pmatrix}$$

$$= \begin{pmatrix}0.846\\0.385\end{pmatrix}$$

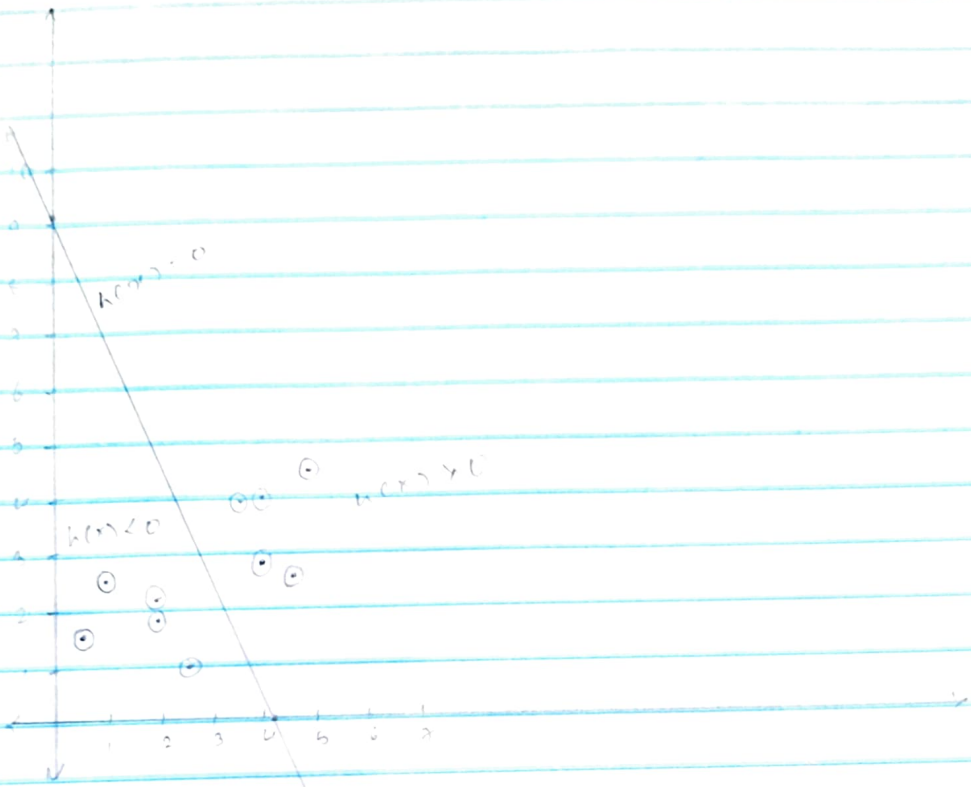
Bias of each support vector.

x_i	$w^T x_i$	$b_i = y_i - w^T x_i$
x_1	4.5005	-3.5005
x_4	2.5000	-3.5000
x_7	4.5010	-3.5010
x_9	2.5005	-3.5005

final bias is average of above bias

$\therefore b = -3.5005$

$$h(x) = \begin{pmatrix} 0.846 \\ 0.385 \end{pmatrix}^T x - 3.5005 = 0$$



All the points where $h(x) < 0$ have value of $y = -1$ and all the points where $h(x) > 0$ have value of $y = 1$.

$h(x) = 0$

$$\therefore \begin{pmatrix} 0.846 \\ 0.385 \end{pmatrix}^T x - 3.5005 = 0$$

$$\begin{pmatrix} 0.846 \\ 0.385 \end{pmatrix}^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 3.5005$$

$\therefore \boxed{x_2 = 9.092}$, if $x_1 = 0$ & for $x_2 = 0$, $\boxed{x_1 = 4.1377}$

2) Distance of x_6 from hyperplane (δ):

$$\delta = y_n = y \frac{h(x)}{\|w\|}$$

Now, $x_6 = \begin{pmatrix} 1.9 \\ 1.9 \end{pmatrix}$

$$h(x_6) = \begin{pmatrix} 0.846 \\ 0.385 \end{pmatrix}^T \begin{pmatrix} 1.9 \\ 1.9 \end{pmatrix} - 3.5005$$

$$= 2.3389 - 3.5005$$

$$= -1.1616 \quad \text{and} \quad \|w\| = \sqrt{7.22}$$

$$\therefore \delta = y_r = y \frac{h(x)}{\|w\|}$$

$$= \frac{(-1)(-1.1616)}{\sqrt{7.22}}$$

$$\boxed{\delta = 0.432}$$

P.T.O.

* Margin of the classifier can be defined as the minimum of distances of a point from the hyperplane here)

$$\delta^* = \min_{x_i} \left\{ y_i \frac{(w^T x_i + b)}{\|w\|} \right\}$$

$$\therefore \delta_{x_1} = 1 \frac{(4.5005 - 3.5005)}{\sqrt{24.41}}$$

$$= 0.2024$$

$$\therefore \delta_{x_2} = 1 \frac{(4.924 - 3.5005)}{\sqrt{32}}$$

$$= 0.2516$$

$$\therefore \delta_{x_3} = \frac{(-1)(1.8085 - 3.5005)}{\sqrt{7.25}}$$

$$= 0.6285$$

$$\therefore \delta_{x_4} = \frac{(-1)(2.5 - 3.5005)}{\sqrt{7.25}}$$

$$= 0.3716$$

$$\therefore \delta_{x_5^*} = \frac{(1)(5.88 - 3.5005)}{\sqrt{44.26}}$$

$$= 0.3573$$

$$\therefore \delta_{x_6} = (-1)\frac{(2.3389 - 3.5005)}{\sqrt{7.22}}$$

$$= 0.4323$$

$$\therefore \delta_{x_7} = (1)\frac{(4.5010 - 3.5005)}{\sqrt{28.25}}$$

$$= 0.1882$$

$$\therefore \delta_{x_8} = (-1)\frac{(1.0005 - 3.5005)}{\sqrt{2.5}}$$

$$= 1.581$$

$$\therefore \delta_{x_9} = (-1)\frac{(2.5005 - 3.5005)}{\sqrt{3.41}}$$

$$= 0.3448$$

$$\therefore \delta_{x_{10}} = (1)\frac{(4.7695 - 3.5005)}{\sqrt{26.5}}$$

$$= 0.2465$$

$$\therefore \delta^* = \min(\delta_{x_i})$$

$$\boxed{\delta^* = 0.1882} \quad (x_7)$$

Given point x_6 and distance is 0.432 for x_6 and so it is not the margin.

3) we classify the point x as follows;

$$\hat{y} = \text{sign}(h(z))$$

$$= \text{sign}(w^T z + b)$$

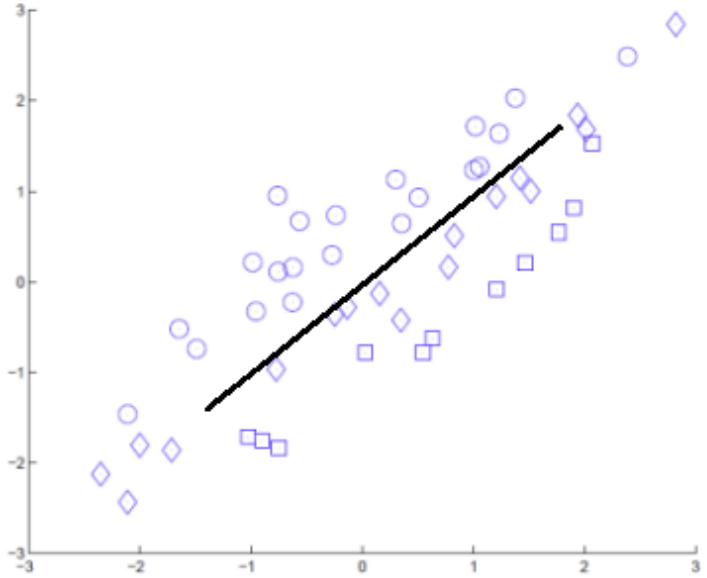
$$= \text{sign}\left[\begin{pmatrix} 0.846 \\ 0.385 \end{pmatrix}^T \begin{pmatrix} 3 \\ 3 \end{pmatrix} - 3.5005\right]$$

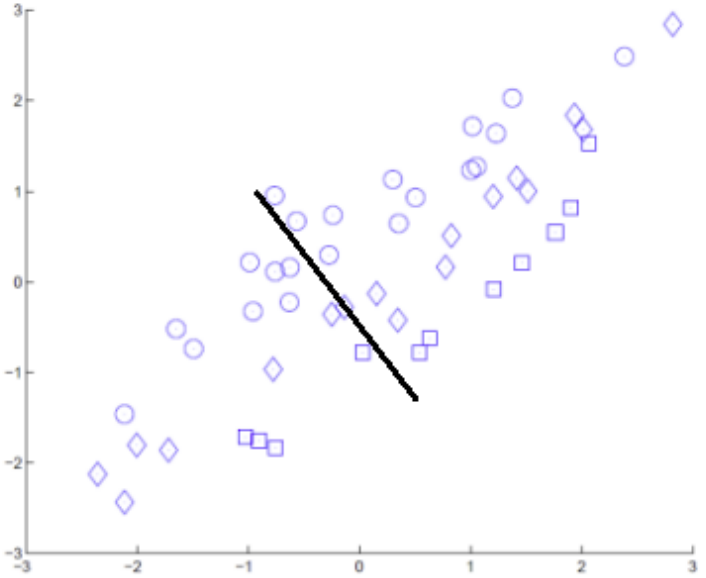
$$= \text{sign}\left[3.693 - 3.5005\right]$$

$$= \text{sign}(0.1925)$$

Now sign function returns +1 if the value of argument is positive and -1 if argument is negative.

→ Here 0.1925 is positive, so we classify the point x as +1.


3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

Problem 1:

Importing Libraries

In [453]:

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import cross_val_score
```

Reading the dataset

In [454]:

```
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
headers = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
iris = pd.read_csv(url, names = headers)
names = iris['class'].unique()
iris.head()
```

Out[454]:

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Since I wasn't sure of what I was supposed to do for this question, I did two things which I thought you might have asked. For the first part I have divided the dataset into pairs of classes and then I have implemented LDA on 67% of the grouped dataset and then tested it against for our test dataset and reported an accuracy score for that. For the second part I have just implemented a simple LDA for all the classes combined and plotted a graph out of the results. ¶

Part 1

Grouping the dataset into pairs

In [455]:

```
grouped = iris.groupby('class')

# t1,t2,t3 contain datasets of class types 1,2,3 respectively
t1 = grouped.get_group(names[0])
t2 = grouped.get_group(names[1])
t3 = grouped.get_group(names[2])

# pairing datasets since we will perform LDA pairwise
one_and_two = t1.append(t2)
two_and_three = t2.append(t3)
one_and_three = t1.append(t3)
```

Defining LDA model

In [456]:

```
lda = LinearDiscriminantAnalysis(n_components = 1)
```

For classes 1 and 2:

Seperating features and the classes

In [457]:

```
X = one_and_two[one_and_two.columns[0:4]]
y = one_and_two[one_and_two.columns[4]]
```

Testing LDA model

In [458]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, r
andom_state=42)

lda.fit(X_train, y_train)
y_pred = lda.predict(X_test)
print('Accuracy score:')
accuracy_score(y_test,y_pred)
```

```
Accuracy score:
```

Out[458]:

```
1.0
```

For classes 2 and 3:

Seperating features and classes

In [459]:

```
X = two_and_three[two_and_three.columns[0:4]]
y = two_and_three[two_and_three.columns[4]]
```

Testing LDA model

In [460]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, r
andom_state=42)

lda.fit(X_train, y_train)
y_pred = lda.predict(X_test)
print('Accuracy score:')
accuracy_score(y_test,y_pred)
```

```
Accuracy score:
```

Out[460]:

```
0.8787878787878788
```

For classes 1 and 3:

Seperating features and classes

In [461]:

```
X = one_and_three[one_and_three.columns[0:4]]
y = one_and_three[one_and_three.columns[4]]
```

Testing LDA model

In [462]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, r
andom_state=42)

lda.fit(X_train, y_train)
y_pred = lda.predict(X_test)
print('Accuracy score:')
accuracy_score(y_test,y_pred)
```

```
Accuracy score:
```

Out[462]:

```
1.0
```

Part 2

Here I just apply simple LDA to the whole dataset and plot the obtained result on a graph to visualize the groupings done by the model.

Seperating features and classes

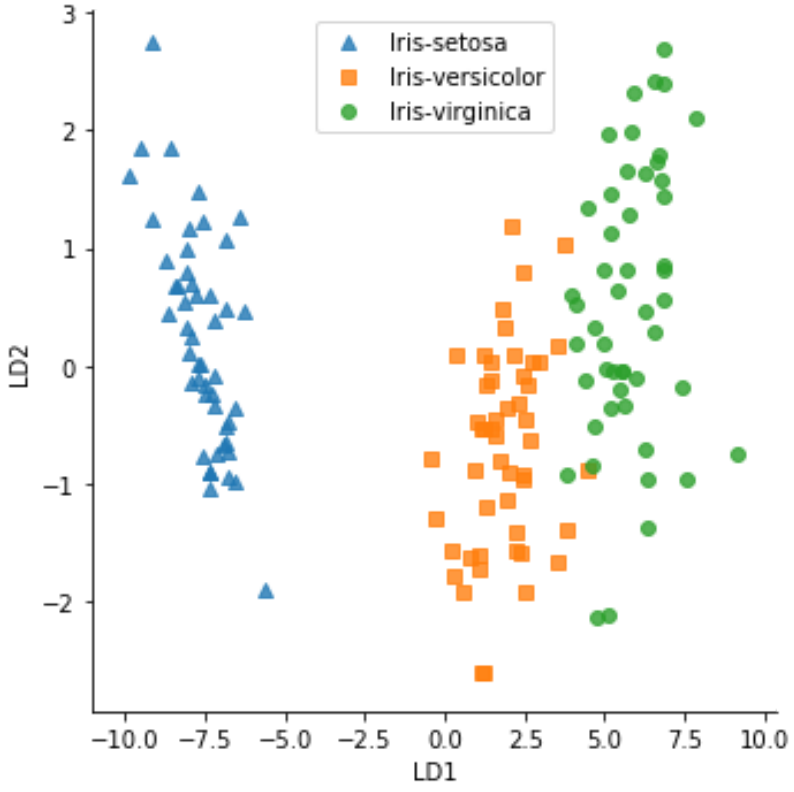
In [463]:

```
X = iris[iris.columns[0:4]]
y = iris[iris.columns[4]]
```

Implementing LDA model

In [464]:

```
lda = LinearDiscriminantAnalysis(n_components = 2)
# passing features as an input and class as an output
X_vals = lda.fit(X,y)
# we know that for LDA, we get two parameters ld1 and ld2, so X_vals contai
ns ld1 and ld2
X_vals = X_vals.transform(X)
# making a final dataframe contain X_vals and corresponding y values
lda_final = pd.DataFrame(X_vals)
lda_final['class'] = y
lda_final.columns=["LD1","LD2","class"]
# plotting the graph LD1 vs LD2
markers = ['^', 's', 'o']
sns.lmplot(x="LD1", y="LD2", data = lda_final, hue = 'class', markers = mar
kers, fit_reg=False, legend=False)
plt.legend(loc='upper center')
plt.show()
```



Problem 2 (2):

Importing Libraries

In [465]:

```
from numpy import log, dot, e
from sklearn.model_selection import KFold
import random
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import load_breast_cancer
```

Reading and scaling the dataset

In [466]:

```
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data'
headers = ['id', 'type', 'mean radius', 'mean texture', 'mean perimeter', 'mean area', 'mean smoothness', 'mean compactness', 'mean concavity', 'mean concave points', 'mean symmetry', 'mean fractal dimension', 'radius error', 'texture error', 'perimeter error', 'area error', 'smoothness error', 'compactness error', 'concavity error', 'concave points error', 'symmetry error', 'fractal dimension error', 'worst radius', 'worst texture', 'worst perimeter', 'worst area', 'worst smoothness', 'worst compactness', 'worst concavity', 'worst concave points', 'worst symmetry', 'worst fractal dimension']
bc = pd.read_csv(url,names = headers)

# Replacing M with 1 and B with 0 for simplicity
bc = bc.replace('M',1)
bc = bc.replace('B',0)

# Converting to array for scaling
temp = bc[bc.columns[1:2]]
temp = temp.to_numpy().reshape(len(temp))
temp2 = bc[bc.columns[2:]]
temp2 = temp2.to_numpy()
```

Defining our custom Logistic Regression with Stochastic and Mini-Batch gradient descent

Stochastic Gradient Descent:

In [467]:

```
class LogisticRS:

    # defining instance variables
    def __init__(self, lr = 0.0001, iters = 9000):
        self.lr = lr
        self.iters = iters
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        # initialize paramenters
        samples, features = X.shape
        # initializing weights to zero
        self.weights = np.zeros(features)
        self.bias = 0
        #for graph plotting purposes
        cost_list = []
        epoch_list = []

        # stochastic gradient descent algorithm
        for i in range(self.iters):
            # picking a random index
            r_index = random.randint(0,samples - 1)
            # getting data for that particular index
            sample_x = X[r_index]
            sample_y = y[r_index]
            # getting value for the logistic regression equation
            linear_model = np.dot(sample_x, self.weights) + self.bias
            # applying sigmoid to the value
            y_pred = self._sigmoid(linear_model)
            # getting derivative
            dw = (1/samples) * np.dot(sample_x.T,y_pred - sample_y)
            db = (1/samples) * np.sum(y_pred - sample_y)
            # updating the weights accordingly
            self.weights -= self.lr*dw
            self.bias -= self.lr*db
            # cost function
            cost = np.square(sample_y-y_pred)

            if i%100==0: # at every 100th iteration record the cost and ite
rs value
                cost_list.append(cost)
                epoch_list.append(i)


        return cost_list,epoch_list

    def predict(self, X):
        # predicting according to the obtained weights
        linear_model = np.dot(X, self.weights) + self.bias
        #applying sigmoid
```

```
        y_pred = self._sigmoid(linear_model)
        # assigns 1 if pred value is more than 0.5 and 0 otherwise
        y_pred_cls = [1 if i > 0.5 else 0 for i in y_pred]
        return y_pred_cls

    def _sigmoid(self, x):
        # simple sigmoid function
        return 1/(1 + np.exp(-x))
```

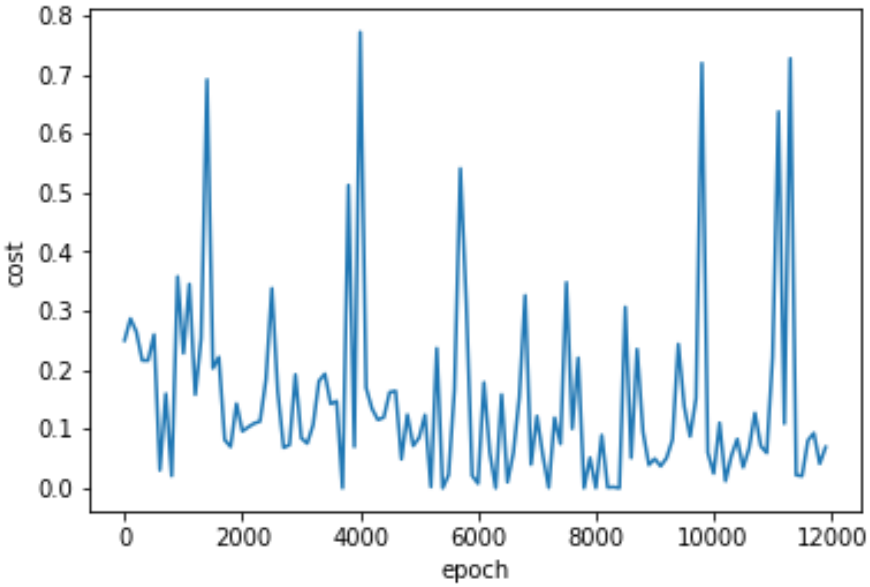
Using the defined model:

In [468]:

```
if __name__ == "__main__":
    # obtaining normal accuracy of the model
    def accuracy(y_true, y_pred):
        accuracy = np.sum(y_true == y_pred) / len(y_true)
        return accuracy

    # temp2 and temp are data and target values
    X, y = temp2,temp
    X_train, X_test, y_train, y_test = train_test_split(temp2, temp, test_s
ize=0.33, random_state=1234)

    # can change lr and iters to obtain different results
    regressor = LogisticRS(lr = 0.0001, iters=12000)
    axisx,axisy = regressor.fit(X_train, y_train)
    predictions = regressor.predict(X_test)

    print("LR classification accuracy:", accuracy(y_test, predictions))

    # Confusion matrix and different scores
    cm = [[0,0],[0,0]]

    for i in range(len(y_test)):
        if(y_test[i] == 0 and predictions[i] == 0):
            cm[0][0] += 1
        if(y_test[i] == 0 and predictions[i] == 1):
            cm[0][1] += 1
        if(y_test[i] == 1 and predictions[i] == 0):
            cm[1][0] += 1
        if(y_test[i] == 1 and predictions[i] == 1):
            cm[1][1] += 1

    # true negative, false positive, false negative, true positive
    tn = cm[0][0]
    fp = cm[0][1]
    fn = cm[1][0]
    tp = cm[1][1]

    precision_score = tp/(fp + tp)
    recall_score = tp/(fn + tp)
    accu_score = (tp + tn)/(tp + fn + tn + fp)
    print('Precision: %.3f' % precision_score)
    print('Recall: %.3f' % recall_score)
    print('Accuracy: %.3f' % accu_score)


    plt.xlabel("epoch")
    plt.ylabel("cost")
    plt.plot(axisy,axisx)

    #cross validation
    kf = KFold(n_splits=3, random_state=None)
```

```
acc_score = []

for train , test in kf.split(X):
    X_train , X_test = X[train,:],X[test,:]
    y_train , y_test = y[train] , y[test]

    regressor.fit(X_train,y_train)
    pred_values = regressor.predict(X_test)

    acc = accuracy(pred_values, y_test)
    acc_score.append(acc)

avg_acc_score = sum(acc_score)/3

print('accuracy of each fold - {}'.format(acc_score))
print('Avg accuracy : {}'.format(avg_acc_score))
```

```
LR classification accuracy: 0.898936170212766
Precision: 0.964
Recall: 0.761
Accuracy: 0.899
accuracy of each fold - [0.8421052631578947, 0.852631578947368
4, 0.91005291005291]
Avg accuracy : 0.8682632507193911
```



Mini-Batch Gradient Descent:

In [469]:

```
class LogisticRM:

    # defining instance variables
    def __init__(self, lr = 0.0001, iters = 100, batch_size = 10):
        self.lr = lr
        self.iters = iters
        self.weights = None
        self.bias = None
        self.batch_size = batch_size

    def fit(self, X, y):
        # initialize paramenters
        samples, features = X.shape
        # initially we set all weights equal to 0 and bias equal to 0
        self.weights = np.zeros(features)
        self.bias = 0
        # for graph plotting purposes
        cost_list = []
        epoch_list = []
        # mini-batch gradient descent algorithm
        for i in range(self.iters):
            # basically just shuffling our dataset
            r_indices = np.random.permutation(samples)
            sample_x = X[r_indices]
            sample_y = y[r_indices]

            # iterating all of the data in small batches
            for j in range(0,samples,self.batch_size):
                Xt = sample_x[j:j+self.batch_size]
                yt = sample_y[j:j+self.batch_size]

                # forming the linear logistic regression equation
                linear_model = np.dot(Xt, self.weights) + self.bias

                # getting the sigmoid values for linear_model
                y_pred = self._sigmoid(linear_model)

                # this is the derivative part
                dw = (1/samples) * np.dot(Xt.T,y_pred - yt)
                db = (1/samples) * np.sum(y_pred - yt)

                # updating the weights
                self.weights -= self.lr*dw
                self.bias -= self.lr*db

                # MSE of costs
                cost = np.mean(np.square(yt-y_pred))

            if i%10==0: # at every 10th iteration record the cost and iters
value
                cost_list.append(cost)
```

```
                epoch_list.append(i)


        return cost_list,epoch_list

    def predict(self, X):
        # for testing the model
        linear_model = np.dot(X, self.weights) + self.bias
        y_pred = self._sigmoid(linear_model)
        # assigns 1 for pred value more than 0.5 and 0 otherwise
        y_pred_cls = [1 if i > 0.5 else 0 for i in y_pred]
        return y_pred_cls

    def _sigmoid(self, x):
        # simple sigmoid function
        return 1/(1 + np.exp(-x))
```

Using the defined model:

In [470]:

```
if __name__ == "__main__":
    # simple accuracy function for predictions
    def accuracy(y_true, y_pred):
        accuracy = np.sum(y_true == y_pred) / len(y_true)
        return accuracy

    # temp2 and temp are data and target variable respectively
    X, y = temp2,temp
    X_train, X_test, y_train, y_test = train_test_split(temp2, temp, test_s
ize=0.33, random_state=1234)

    # for simplicity purposes don't enter batch_size greater than the sampl
e size
    regressor = LogisticRM(lr = 0.0001, iters=500, batch_size = 20)
    axisx,axisy = regressor.fit(X_train, y_train)
    predictions = regressor.predict(X_test)

    print("LR classification accuracy:", accuracy(y_test, predictions))


    plt.xlabel("epoch")
    plt.ylabel("cost")
    plt.plot(axisy,axisx)

    # Confusion matrix and different scores
    cm = [[0,0],[0,0]]

    for i in range(len(y_test)):
        if(y_test[i] == 0 and predictions[i] == 0):
            cm[0][0] += 1
        if(y_test[i] == 0 and predictions[i] == 1):
            cm[0][1] += 1
        if(y_test[i] == 1 and predictions[i] == 0):
            cm[1][0] += 1
        if(y_test[i] == 1 and predictions[i] == 1):
            cm[1][1] += 1

    # true negative, false positive, false negative, true positive
    tn = cm[0][0]
    fp = cm[0][1]
    fn = cm[1][0]
    tp = cm[1][1]

    precision_score = tp/(fp + tp)
    recall_score = tp/(fn + tp)
    accu_score = (tp + tn)/(tp + fn + tn + fp)
    print('Precision: %.3f' % precision_score)
    print('Recall: %.3f' % recall_score)
    print('Accuracy: %.3f' % accu_score)
```

```
    # cross validation
    kf = KFold(n_splits=3, random_state=None)

    acc_score = []

    for train , test in kf.split(X):
        X_train , X_test = X[train,:],X[test,:]
        y_train , y_test = y[train] , y[test]

        regressor.fit(X_train,y_train)
        pred_values = regressor.predict(X_test)

        acc = accuracy(pred_values , y_test)
        acc_score.append(acc)

    avg_acc_score = sum(acc_score)/3

    print('accuracy of each fold - {}'.format(acc_score))
    print('Avg accuracy : {}'.format(avg_acc_score))
```

```
LR classification accuracy: 0.9202127659574468
Precision: 0.952
Recall: 0.831
Accuracy: 0.920
accuracy of each fold - [0.8894736842105263, 0.921052631578947
3, 0.9259259259259259]
Avg accuracy : 0.9121507472384666
```