1. (10pts) Problem 2.1 in LFD

1. $\sqrt{\frac{1}{2N}\ln\frac{2M}{\delta}} \le \varepsilon \Leftrightarrow N \ge \frac{1}{2\varepsilon^2}\ln\frac{2M}{\delta}$

a) For $M=1$ and $\delta = 0.03$, to have $\varepsilon \le 0.05$.

$$N \ge \frac{1}{2\cdot(0.05)^2}\ln\frac{2}{0.03}. \quad \boxed{= 839.9410.}$$

b) For $M=100$ and $\delta=0.03$, $\varepsilon \le 0.05$ we need.

$$N \ge \frac{1}{2(0.05)^2}\ln\frac{2\cdot 100}{0.03}$$

$$\boxed{N \ge 1760.9750.}$$

c) For $M = 10000$ & $\delta = 0.03$, $\varepsilon \le 0.05$ we need.

$$N \ge \frac{1}{2(0.05)^2}\ln\frac{2\cdot 10000}{0.03}$$

$$\boxed{N \ge 2682.00909.}$$

2. (10pts) Problem 2.3 in LFD

2. a) Growth function for positive rays is equal to $N+1$.
If dichotomies added by negative rays, we get $N-1$ new dichotomies.
So, total.

$$\boxed{m_{\mathcal{H}}(N) = 2N}$$

largest value of N $m_{\mathcal{H}}(N) = 2^N$ is $2 (m_{\mathcal{H}}(3) = 6)$.
We have $d_{vc} = 2$.

b) Growth function for positive intervals is equal to

$$N^2/2 + N/2 + 1.$$

Adding new dichotomies generated by negative intervals, we get $N-2$ new, only if $N > 1$.
If $N = 1$ generate two dichotomies with only positive intervals.

$$m_{\mathcal{H}}(N) = \frac{N^2}{2} + \frac{3N}{2} - 1.$$

if $N > 1$.

and $m_{\mathcal{H}}(N) = 2$. if $N = 1$.

So, $m_H(N) = 2^N$, the largest value of $N = 3$.
$d_{vc} = 3$.

c) $\phi : (x_1, x_2, \ldots, x_d) \rightarrow r = \sqrt{x_1^2 + \ldots + x_d^2}$.
Concentric circles in R^d is same as the problem of positive intervals in R.

$$m_{\mathcal{H}}(N) = \frac{N^2}{2} + \frac{N}{2} + 1.$$

dependent of d.
Largest value of N $m_H(N) = 2^N$ is 2.
$d_{vc} = 2$.

3. (10pts) Problem 2.8 in LFD

3) Two cases of growth function:
$d_{vc} = +\infty$ and $m_H(N) = 2^N$ for all N
and d_{vc} is finite & $m_H(N)$ is bounded by $N^{d_{vc}}+1$.
If $m_H(N) = 1+N$, $d_{vc} = 1$. So it must be bounded by $N+1$ for all N.

So, $m_H(N) = N+1$ is a growth function

If $m_H(N) = 1+N+\frac{N(N-1)}{2}$, we have $d_{vc} = 2$.
So, it is bound by N^2+1 for all N.
Thus $m_H(N) = 1+N+\frac{N(N-1)}{2}$ is a growth function
And $m_H(N) = 2^N$ is a growth function ($d_{vc} = +\infty$).

If $m_H(N) = 2^{(\sqrt{N})}$, we have $d_{vc} = 1$.
It must be bounded by $N+1$ for all N, which isn't true (for all N). Thus, it is a possible growth function

If $m_H(N) = 2^{\lfloor N/2 \rfloor}$, we have $d_{vc} = 0$. It must be bounded by $N^0+1 = 2$. for all N, which isn't true for all N
Thus, it isn't a possible growth function.

4. (10pts) Problem 2.12 in LFD

4) $N \geq \frac{8}{(0.05)^2} \ln\left(\frac{4[(2N)^{10}+1]}{0.05}\right)$, $d_{vc}=10$, $\varepsilon = 0.05$ & $\delta = 0.05$

Assume $N = 1000$.

$$N \geq \frac{8}{(0.05)^2} \ln\left(\frac{4[(2\cdot 1000)^{10}+1]}{0.05}\right)$$

$$\approx 2.58 \times 10^5$$

Trying new value of $N = 2.58\times 10^5$ the $N \approx 4.52 \times 10^5$

5. (10pts) Problem 2.22 in LFD

Q5 $E[E_{out}(\vec{x})] = E_{D,\varepsilon}[(g(\vec{x}) - f(\vec{x}) - \varepsilon)^2]$

$$= E_{D,\varepsilon}[(g(\vec{x}) - f(\vec{x}))^2 - 2(g(\vec{x})\cdot f(\vec{x}))\varepsilon + \varepsilon]$$

approximation of

$$\underbrace{2[E_{D,\varepsilon}[g(\vec{x}) - f(\vec{x})]]}_{0}\varepsilon$$

target function.

$E(E_{out}(\vec{x})) = \sigma^2 + \text{bias} + \text{var}$.

6. (10pts) Prove that selecting the hypothesis h that maximizes the likelihood $\Pi(y_n|x_n)_{n=1}^{N}$ is equivalent to minimizing the cross-entropy error
$E_i(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}\ln(1+e^{-y_n \boldsymbol{w}^T x_n})$

Q.6.

$$\max_{w} \prod_{n=1}^{N} P(y_n|\vec{x}_n).$$

$$\rightarrow \max_{w} \log\left(\prod_{n=1}^{N} \theta(y_n w^T x_n)\right)$$

$$\min_{w} -\log\left(\prod_{n=1}^{N} \theta(y_n w^T x_n)\right)$$

$$\min_{w} -\sum_{n=1}^{N} \log(\theta(y_n w^T x_n))$$

$$\min_{w} \sum_{n=1}^{N} \log\left(\frac{1}{\theta(y_n w^T x_n)}\right)$$

$$\min_{w} \sum_{n=1}^{N} \log\left(1+e^{-y_n w^T x_n}\right).$$

7. (10pts) Derive the gradient of the in-sample error $\nabla E_i(\mathbf{w}(t))$ used in the gradient descent algorithm.

Q.7. $\Delta E_{in} = E[w(0) + \eta \vec{v}]$

→ Using the Taylor series

$$E_{in}(x) = E_{in}(w_0) + (x - w_0)^T \nabla E_{in}(w_0) + O(\|x - w_0\|_2^2)$$

So, evaluating the Ein at $w_0 + \eta \hat{v}$ for some unit vector $\hat{v}$, gives:

$$E_{in}(w_0 + \eta \hat{v}) = E_{in}(w_0) + (\eta \hat{v})^T \nabla E_{in}(w_0) + O(\|\eta \hat{v}\|_2^2)$$

$$= E_{in}(w_0) + \eta \nabla E_{in}(w_0)^T \hat{v} + O(\eta)^2 \quad -(3)$$

The difference is:-

$$\Delta E_{in} = E_{in}(w_0 + \eta \hat{v}) - E_{in}(w_0)$$

$$= \eta \nabla E_{in}(w_0)^T \hat{v} + O(\eta)^2$$ as required.

8. (10pts) Exercise 3.6

Q. $P(y|x)$ captured by hypothesis $h(x)$; that likelihood ~~would~~ would be

$$P(y|x) = \begin{cases} h(x) & \text{for } y = +1 \\ 1-h(x) & \text{for } y = -1 \end{cases}$$

with noisy target.

Minimize $E_{in}(w)$ that is in sample error only with respect to w, so

$$E_{in} = \frac{1}{N} \sum_{n=1}^{N} \ln\left(1 + e^{-y_n w^T x_n}\right) - ①$$

where $w^T x_n = h(x_n)$ - ②

substituting ② in ① we get,

$$E_{in} = \frac{1}{N} \sum_{n=1}^{N} \ln\left(1 + e^{-(y_n) h(x_n)}\right) - ③$$

from eq^n ③ we can say that $h(x_n)$ is a very positive number and y_n is positive.

i.e. whatever output probability we prove / predict happens to be true.

So, $e^{-(y_n) h(x_n)}$ becomes -ve power of e & error is very small.

→ if $h(x_n)$ is a -ve & y_n is +ve, then $e^{-y_n w^T x}$ becomes +ve

We didn't predict it though it happens so, $e^{-y_n w^T x_n}$ become +ve e^{+} which indicates high error.

→ Thus, the maximum likelihood would be achieved when $e^{-w^T x_n}$ minimizes or reduces to $\frac{1}{h(x)}$ → which will reduce if $h(x)$ is high.

if $y_n = \pm 1$, we can prove that

$$E_{in}(w) = \sum_{n=1}^{N} |y_n = +1| \ln \frac{1}{h(x_n)} + |y_n = -1| \ln \frac{1}{1-h(x_n)} \text{ is minimum}$$

Thus, proved.

b) for $h(x) = \theta(w^T x)$.

$\theta = \dfrac{e^s}{1+e^s}$ or $\dfrac{1}{1+e^{-s}}$,

Thus, dividing with e^s $h(x)$ becomes large and error in that case becomes small.
So, it is equivalent to minimizing in 1, when we convert $e^{w^T x_n}$ to $\ln \frac{1}{}$ to make the error smaller for larger $h(x_n)$. $h(x_n)$

9. (10pts) Exercise 3.13

(a)

Parabola

$(x_1 - 3)^2 + x_2 = 1.$

$x_1^2 - 6x_1 + 9 + x_2 - 1 = 0.$

$x_1^2 - 6x_1 + x_2 + 8 = 0$

$w = (1, 0, -6, 1, 0, 0)$ and intercept 8.

(b)

$\rightarrow$ Circle $\rightarrow (x_1 - 3)^2 + (x_2 - 4)^2 = 1.$

$(x_1 - 3)^2 + (x_2 - 4)^2 - (1)^2 = 0.$

$x_1^2 - 6x_1 + 9 + x_2^2 - 8x_2 + 16 - 1 = 0.$

$x_1^2 - 6x_1 + x_2^2 - 8x_2 + 24 = 0.$

$\boxed{w = (1, -6, -8, 1, 0, 1). \text{ and intercept } 24.}$

(c)

Ellipse equation $= c(x_1-a)^2 + d(x_2-b)^2 - 1 = 0$.

Here eq^n is $2(x_1-3)^2 + (x_2-4)^2 = 1$

$= 2x_1^2 + x_2^2 - 2\times 3\times 2x_1 - 2\times 4\times 1\, x_2 + ((3)^2 2 + (4)^2 1 - 1)$

$= 2x_1^2 + x_2^2 - 12x_1 - 8x_2 + (18+16-1)$.

$= 2x_1^2 + x_2^2 - 12x_1 - 8x_2 + 33$

$\cancel{(1, -12, -8, 2,}$ $\boxed{(1, -12, -8, 2, 0, 1) \text{ and intercept } 33.}$

10. (20pts) Problem 3.1 in LFD. You can use “LFD Problem 3_1.ipynb” as the start point to generate the data. Feel free to write your own code to generate the data.

Code:

```
from IPython.display import Image
from IPython.display import display
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# inline plotting instead of popping out
%matplotlib inline

# load utility classes/functions that has been taught in
previous labs
# e.g., plot_decision_regions()
import os, sys
module_path = os.path.abspath(os.path.join('.'))
sys.path.append(module_path)
from Lib import *
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_moons

def versiontuple(version):
  return tuple(map(int, (version.split("."))))

n_data_points = 2000

rad = 10
thk = 5
sep = 5
c1 = np.array([(rad+thk)/2, -sep/2])
c2 =  np.array([-(rad+thk)/2, sep/2])
r1 = np.random.rand(n_data_points)*thk+rad
a1 = np.random.rand(n_data_points)*np.pi
```

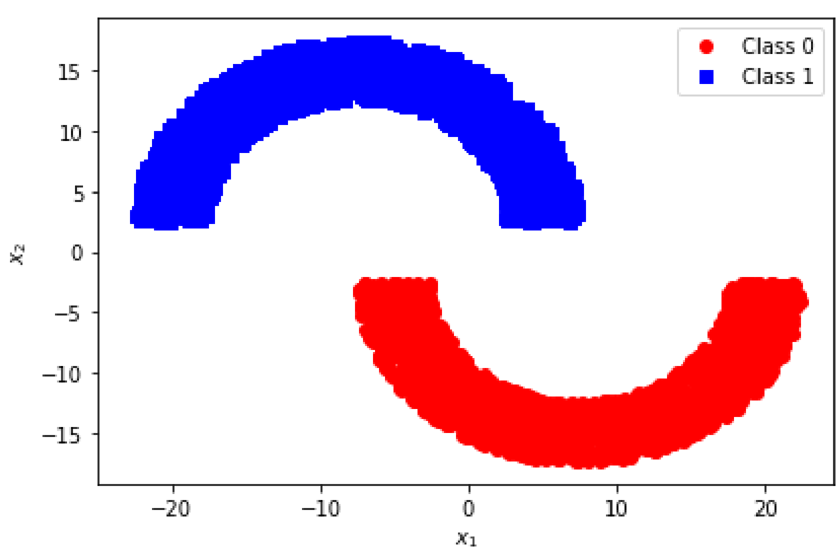
```
r2 = np.random.rand(n_data_points)*thk+rad
a2 = np.random.rand(n_data_points)*np.pi+np.pi

# In order to plot it we convert it to cartesian:
p1 = np.array((r1*np.cos(a1), r1*np.sin(a1)))
p2 = np.array((r2*np.cos(a2), r2*np.sin(a2)))
x1, y1 = (p1[0] - c1[0], p1[1] - c1[1])
x2, y2 = (p2[0] - c2[0], p2[1] - c2[1])

#ones
x3 = x1.reshape(n_data_points,-1)
y3 = y1.reshape(n_data_points,-1)
x3 = np.concatenate((x3,y3),axis=1)
y3 = np.full((1, n_data_points), 1, dtype=int)[0]

#zeros
x4 = x2.reshape(n_data_points,-1)
y4 = y2.reshape(n_data_points,-1)
x4 = np.concatenate((x4,y4),axis=1)
X = np.concatenate((x3,x4),axis=0)
y4 = np.full((1, n_data_points), 0, dtype=int)[0]
y = np.concatenate((y3,y4),axis=0)

plt.scatter(X[y == 0, 0], X[y == 0, 1],
            c='r', marker='o', label='Class 0')
plt.scatter(X[y == 1, 0], X[y == 1, 1],
            c='b', marker='s', label='Class 1')

# plt.xlim(X[:, 0].min()-1, X[:, 0].max()+1)
# plt.ylim(X[:, 1].min()-1, X[:, 1].max()+1)
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.legend(loc='best')
plt.tight_layout()
#plt.savefig('./output/fig-two-moon.png', dpi=300)
plt.show()

X_train, X_test, y_train, y_test = train_test_split(
```

```
    X, y, test_size=0.2, random_state=1)

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))

ppn = Perceptron(max_iter=1000, eta0=0.1, random_state=0)
ppn.fit(X_train_std, y_train)
y_pred = ppn.predict(X_test_std)
print('[Perceptron]')
print('Misclassified samples: %d' % (y_test != y_pred).sum())
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))

def plot_decision_regions(X, y, classifier, test_idx=None,
resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max,
resolution),
                           np.arange(x2_min, x2_max,
resolution))
    Z = classifier.predict(np.array([xx1.ravel(),
xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
```

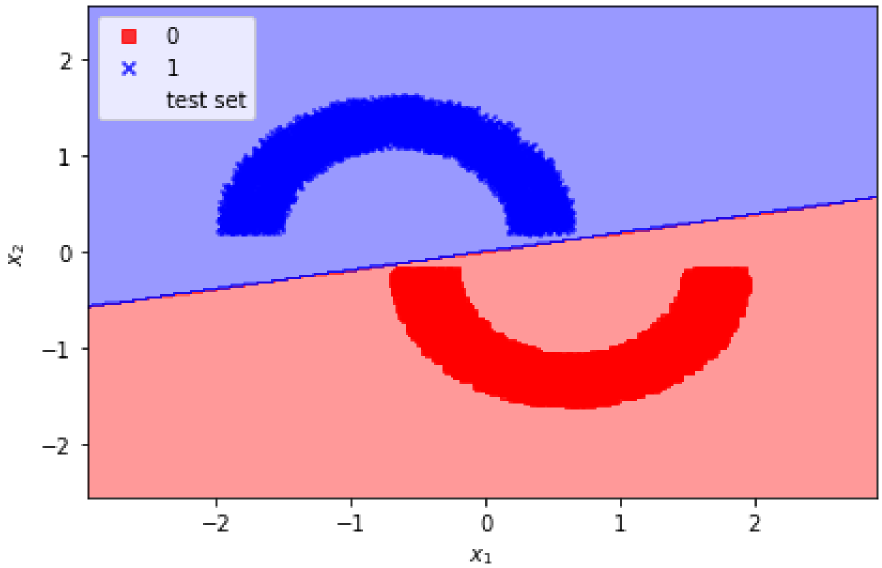
```
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=cmap(idx),
                    marker=markers[idx], label=cl)

    # highlight test samples
    if test_idx:
        # plot all samples
        if not versiontuple(np.__version__) >=
versiontuple('1.9.0'):
            X_test, y_test = X[list(test_idx), :],
y[list(test_idx)]
            warnings.warn('Please update to NumPy 1.9.0 or
newer')
        else:
            X_test, y_test = X[test_idx, :], y[test_idx]

        plt.scatter(X_test[:, 0],
                    X_test[:, 1],
                    c='',
                    alpha=1.0,
                    linewidths=1,
                    marker='o',
                    s=55, label='test set')
# plot decision regions for Perceptron
plot_decision_regions(X_combined_std, y_combined,
                      classifier=ppn,
                      test_idx=range(y_train.size,
                                     y_train.size +
y_test.size))
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.legend(loc='upper left')
plt.tight_layout()
#plt.savefig('./output/fig-two-moon-perceptron-boundray.png',
dpi=300)
plt.show()

lr = LogisticRegression(C=1000.0, random_state=0)
lr.fit(X_train_std, y_train)
```

```
y_pred = lr.predict(X_test_std)
print('[Logistic regression]')
print('Misclassified samples: %d' % (y_test != y_pred).sum())
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))

# plot decision regions for LogisticRegression
plot_decision_regions(X_combined_std, y_combined,
                      classifier=lr,
                      test_idx=range(y_train.size,
                                     y_train.size +
y_test.size))
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.legend(loc='upper left')
plt.tight_layout()
#plt.savefig('./output/fig-two-moon-logistic-regression-
boundray.png', dpi=300)
plt.show()
```

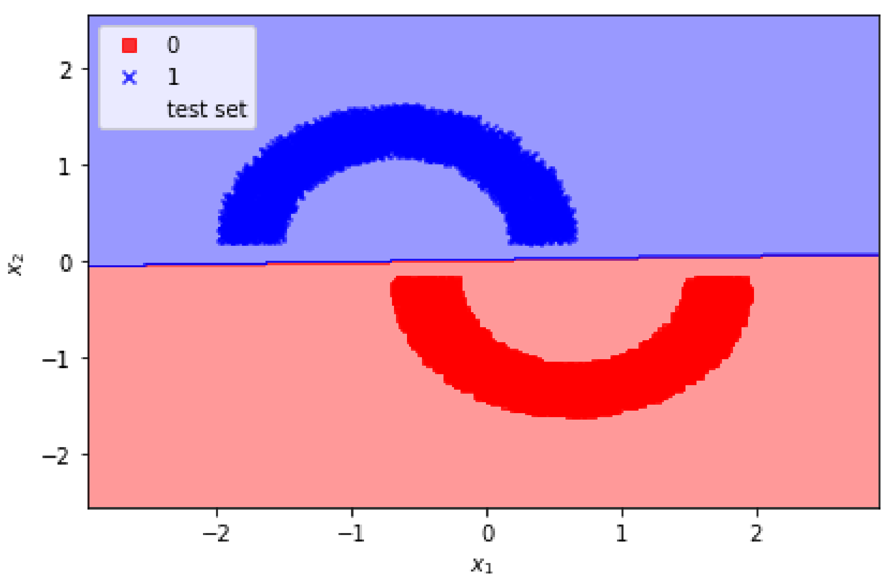
Output:




```
[Perceptron]
Misclassified samples: 0
Accuracy: 1.00
```




```
[Logistic regression]
Misclassified samples: 0
Accuracy: 1.00
```



We can also use linear regression for such classification. Alternatively, linear regression weights w_{lin} are an approximate solution for the perceptron model.

11. (10pts) Problem 3.2 in LFD

```
from IPython.display import Image
from IPython.display import display
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

from scipy import stats
from pylab import *

# inline plotting instead of popping out
%matplotlib inline

import os, sys
module_path = os.path.abspath(os.path.join('.'))
sys.path.append(module_path)
from Lib import *
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_moons

def versiontuple(version):
  return tuple(map(int, (version.split("."))))


n_data_points = 2000

rad = 10
thk = 5
sep = 0.2
c1 = np.array([(rad+thk)/2, sep/2])
c2 =  np.array([-(rad+thk)/2, -sep/2])
r1 = np.random.rand(n_data_points)*thk+rad
a1 = np.random.rand(n_data_points)*np.pi
```

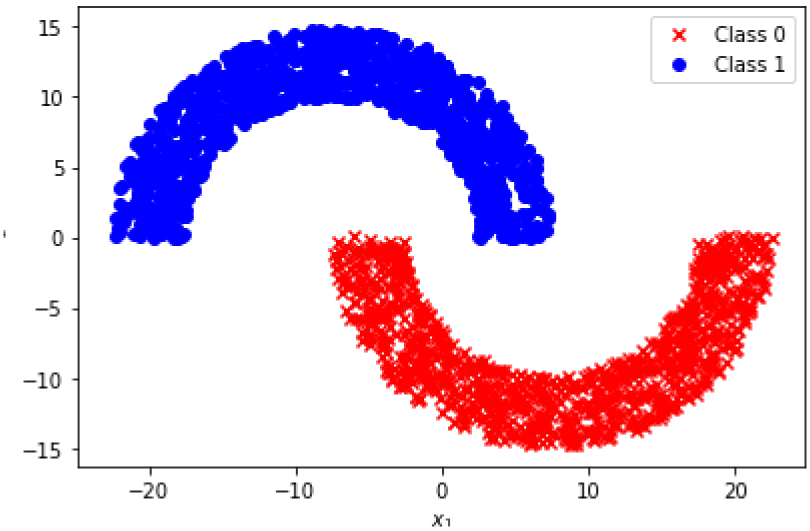
```
r2 = np.random.rand(n_data_points)*thk+rad
a2 = np.random.rand(n_data_points)*np.pi+np.pi

# In order to plot it we convert it to cartesian:
p1 = np.array((r1*np.cos(a1), r1*np.sin(a1)))
p2 = np.array((r2*np.cos(a2), r2*np.sin(a2)))
x1, y1 = (p1[0] - c1[0], p1[1] - c1[1])
x2, y2 = (p2[0] - c2[0], p2[1] - c2[1])

#ones
x3 = x1.reshape(n_data_points,-1)
y3 = y1.reshape(n_data_points,-1)
x3 = np.concatenate((x3,y3),axis=1)
y3 = np.full((1, n_data_points), 1, dtype=int)[0]

#zeros
x4 = x2.reshape(n_data_points,-1)
y4 = y2.reshape(n_data_points,-1)
x4 = np.concatenate((x4,y4),axis=1)
X = np.concatenate((x3,x4),axis=0)
y4 = np.full((1, n_data_points), 0, dtype=int)[0]
y = np.concatenate((y3,y4),axis=0)

plt.scatter(X[y == 0, 0], X[y == 0, 1],
            c='r', marker='x', label='Class 0')
plt.scatter(X[y == 1, 0], X[y == 1, 1],
            c='b', marker='o', label='Class 1')



plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.legend(loc='best')
plt.tight_layout()

plt.show()



X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=1)
```

```
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))

ppn = Perceptron(tol=0.0001,eta0=0.1, random_state=0)
ppn.fit(X_train_std, y_train)
y_pred = ppn.predict(X_test_std)
print('[Perceptron]')
print('Misclassified samples: %d' % (y_test != y_pred).sum())
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
print(ppn.n_iter_)
def plot_decision_regions(X, y, classifier, test_idx=None,
resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max,
resolution),
                           np.arange(x2_min, x2_max,
resolution))
    Z = classifier.predict(np.array([xx1.ravel(),
xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
```

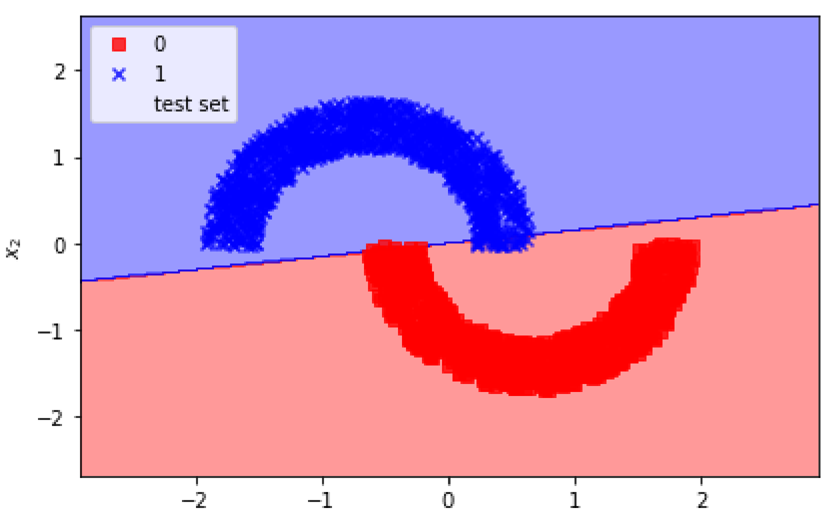
```
                    alpha=0.8, c=cmap(idx),
                    marker=markers[idx], label=cl)

    # highlight test samples
    if test_idx:
        # plot all samples
        if not versiontuple(np.__version__) >=
versiontuple('1.9.0'):
            X_test, y_test = X[list(test_idx), :],
y[list(test_idx)]
            warnings.warn('Please update to NumPy 1.9.0 or
newer')
        else:
            X_test, y_test = X[test_idx, :], y[test_idx]

        plt.scatter(X_test[:, 0],
                    X_test[:, 1],
                    c='',
                    alpha=1.0,
                    linewidths=1,
                    marker='o',
                    s=55, label='test set')
# plot decision regions for Perceptron
plot_decision_regions(X_combined_std, y_combined,
                      classifier=ppn,
                      test_idx=range(y_train.size,
                                     y_train.size +
y_test.size))
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.legend(loc='upper left')
plt.tight_layout()

plt.show()
Sep =
np.array([0.2,0.4,0.6,0.8,1.4,1.6,1.8,2,2.2,2.4,2.6,2.8,3.2,3.4,
3.6,3.8,4,4.2,4.4,4.6,4.8,5])
Iterations =
np.array([0.0,25,180,27,110,60,58,25,60,20,25,5,20,27,40,8,30,5,
6,5,35,24])
```

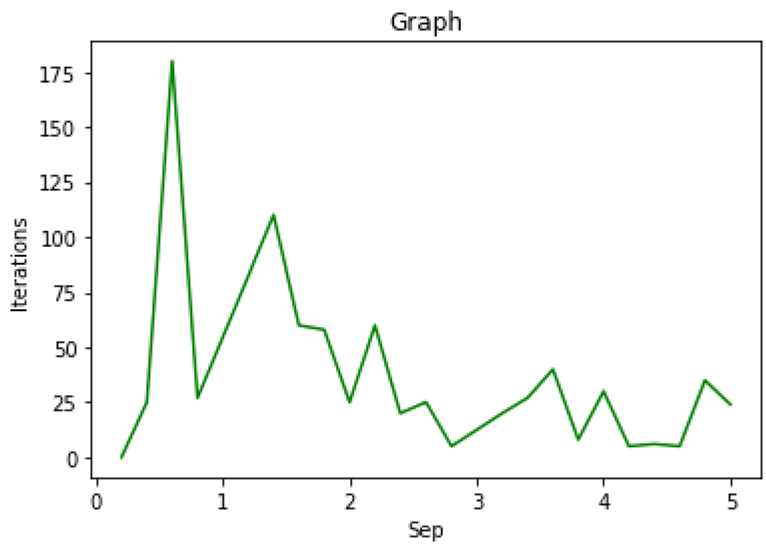
```
plt.plot(Sep,Iterations, color='g')

plt.xlabel('Sep')
plt.ylabel('Iterations')
plt.title('Graph')
plt.show()
```

Output:




```
[Perceptron]
Misclassified samples: 4
Accuracy: 0.99
4
```





The number of iterations tends to decrease when *sep* increases. This can be easily verified by theoretical results as done previously. To prove this here is the plot of “sep versus the maximum-number-of-iterations” in the graph above.

12. (20pts) Problem 3.3 in LFD

Code:

```
from IPython.display import Image
from IPython.display import display
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap


# inline plotting instead of popping out
%matplotlib inline


# load utility classes/functions that has been taught in
previous labs
# e.g., plot_decision_regions()
import os, sys
module_path = os.path.abspath(os.path.join('.'))
sys.path.append(module_path)
from Lib import *
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_moons


def versiontuple(version):
```

```
  return tuple(map(int, (version.split("."))))
n_data_points = 2000
rad = 10
thk = 5
sep = -5
c1 = np.array([(rad+thk)/2, -sep/2])
c2 =  np.array([-(rad+thk)/2, sep/2])
r1 = np.random.rand(n_data_points)*thk+rad
a1 = np.random.rand(n_data_points)*np.pi

r2 = np.random.rand(n_data_points)*thk+rad
a2 = np.random.rand(n_data_points)*np.pi+np.pi

# In order to plot it we convert it to cartesian:
p1 = np.array((r1*np.cos(a1), r1*np.sin(a1)))
p2 = np.array((r2*np.cos(a2), r2*np.sin(a2)))
x1, y1 = (p1[0] - c1[0], p1[1] - c1[1])
x2, y2 = (p2[0] - c2[0], p2[1] - c2[1])

#ones
x3 = x1.reshape(n_data_points,-1)
y3 = y1.reshape(n_data_points,-1)
x3 = np.concatenate((x3,y3),axis=1)
y3 = np.full((1, n_data_points), 1, dtype=int)[0]

#zeros
```

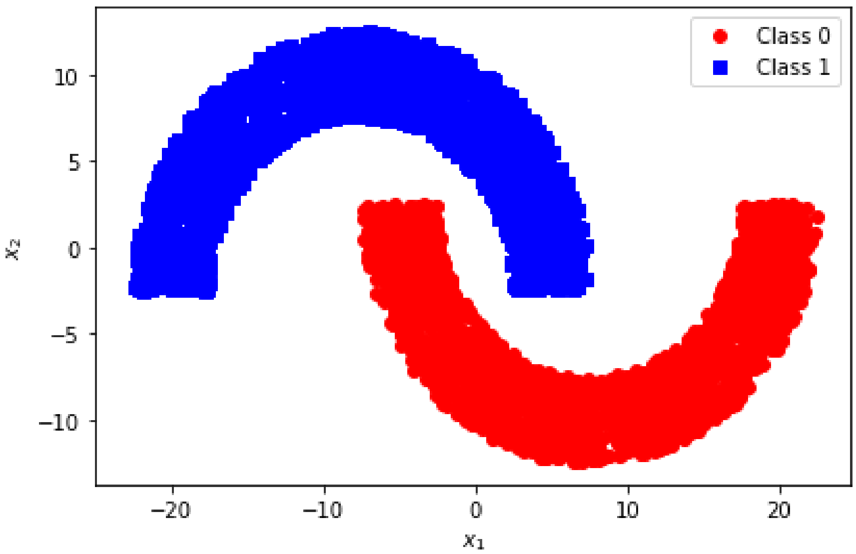
```
x4 = x2.reshape(n_data_points,-1)
y4 = y2.reshape(n_data_points,-1)
x4 = np.concatenate((x4,y4),axis=1)
X = np.concatenate((x3,x4),axis=0)
y4 = np.full((1, n_data_points), 0, dtype=int)[0]
y = np.concatenate((y3,y4),axis=0)

plt.scatter(X[y == 0, 0], X[y == 0, 1],
            c='r', marker='o', label='Class 0')
plt.scatter(X[y == 1, 0], X[y == 1, 1],
            c='b', marker='s', label='Class 1')

# plt.xlim(X[:, 0].min()-1, X[:, 0].max()+1)
# plt.ylim(X[:, 1].min()-1, X[:, 1].max()+1)
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.legend(loc='best')
plt.tight_layout()
#plt.savefig('./output/fig-two-moon.png', dpi=300)
plt.show()

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=1)

sc = StandardScaler()
sc.fit(X_train)
```

```
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))

ppn = Perceptron(max_iter=100000, eta0=0.1, random_state=0)
ppn.fit(X_train_std, y_train)
y_pred = ppn.predict(X_test_std)
print('[Perceptron]')
print('Misclassified samples: %d' % (y_test != y_pred).sum())
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))

def plot_decision_regions(X, y, classifier, test_idx=None,
resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max,
resolution),
```

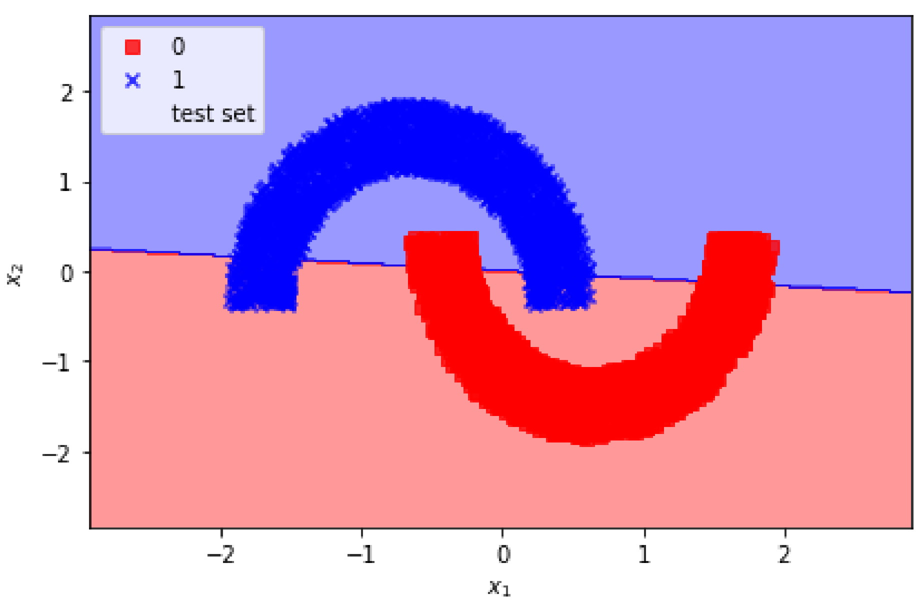
```
                           np.arange(x2_min, x2_max,
resolution))
    Z = classifier.predict(np.array([xx1.ravel(),
xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=cmap(idx),
                    marker=markers[idx], label=cl)

    # highlight test samples
    if test_idx:
        # plot all samples
        if not versiontuple(np.__version__) >=
versiontuple('1.9.0'):
            X_test, y_test = X[list(test_idx), :],
y[list(test_idx)]
            warnings.warn('Please update to NumPy 1.9.0 or
newer')
        else:
            X_test, y_test = X[test_idx, :], y[test_idx]

        plt.scatter(X_test[:, 0],
                    X_test[:, 1],
```

```
                    c='',
                    alpha=1.0,
                    linewidths=1,
                    marker='o',
                    s=55, label='test set')
# plot decision regions for Perceptron
plot_decision_regions(X_combined_std, y_combined,
                      classifier=ppn,
                      test_idx=range(y_train.size,
                                     y_train.size +
y_test.size))
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.legend(loc='upper left')
plt.tight_layout()
#plt.savefig('./output/fig-two-moon-perceptron-boundray.png',
dpi=300)
plt.show()
```

Output:




```
[Perceptron]
Misclassified samples: 107
Accuracy: 0.87
```



a. If we run PLA on these examples, it will never stop updating if we do not specify the maximum iterations.
b. If we run the pocket algorithm for 100000 iterations and we plot *Ein* versus the iteration number *t* for *t* = 1 *to 100. Ein* monotonously decreases opposite to what would happen if we had used the PLA.

In terms of computation time the linear regression algorithm is clearly better than the pocket algorithm. When we consider the quality of the solution, the pocket algorithm has a (final) *Ein* of 0.087, and the linear regression algorithm has a *Ein* of 0.0995. So, in terms of quality of the solution, the pocket algorithm is a little better than the linear regression algorithm.

In most of the cases, regarding the quality of the solution, the linear regression algorithm is also better than the pocket algorithm.

13. (10pts) Problem 3.16 in LFD

Q.13. a) Expected cost:

$$\text{cost(accept)} = 0 \cdot P[y=+1|x] + Ca \cdot P[y=-1|x]$$
$$= Ca(1-g(x))$$

and

$$\text{cost(reject)} = Cr \cdot P[y=+1|x] + 0 \cdot P[y=-1|x]$$
$$= Cr\, g(x).$$

b)

$$\text{cost(accept)} = \text{cost(reject)}.$$

$$Ca(1-g(x)) = Cr \cdot g(x).$$

$$g(x) = \frac{Ca}{Ca+Cr}.$$

Thus,

threshold is $K = \frac{Ca}{Ca+Cr}.$

c) For supermarket, we have $k = 1/11$, to avoid false rejects

For CIA, $K = 1000/1001$ to avoid false accepts

so with $k >>$ we only accept $g(x) \geq k$ close to 1.