# Supervised Logistic Regression for Classification

## 0. Import library

```
# Import libraries

# math library
import numpy as np

# visualization library
%matplotlib inline
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('png2x','pdf')
import matplotlib.pyplot as plt

# machine learning library
from sklearn.linear_model import LogisticRegression

# 3d visualization
from mpl_toolkits.mplot3d import axes3d

# computational time
import time

```

## 1. Load dataset

The data features $x_i = (x_{i(1)}, x_{i(2)})$ represent 2 exam grades $x_{i(1)}$ and $x_{i(2)}$ for each student $i$.

The data label $y_i$ indicates if the student $i$ was admitted (value is 1) or rejected (value is 0).

```
# import data with numpy
data = np.loadtxt('/content/drive/My Drive/Colab Notebooks/MachineLearningProject/04/dataset.txt'

# number of training data
n = data.shape[0]
print('Number of training data=',n)
```

```
Number of training data= 100
```

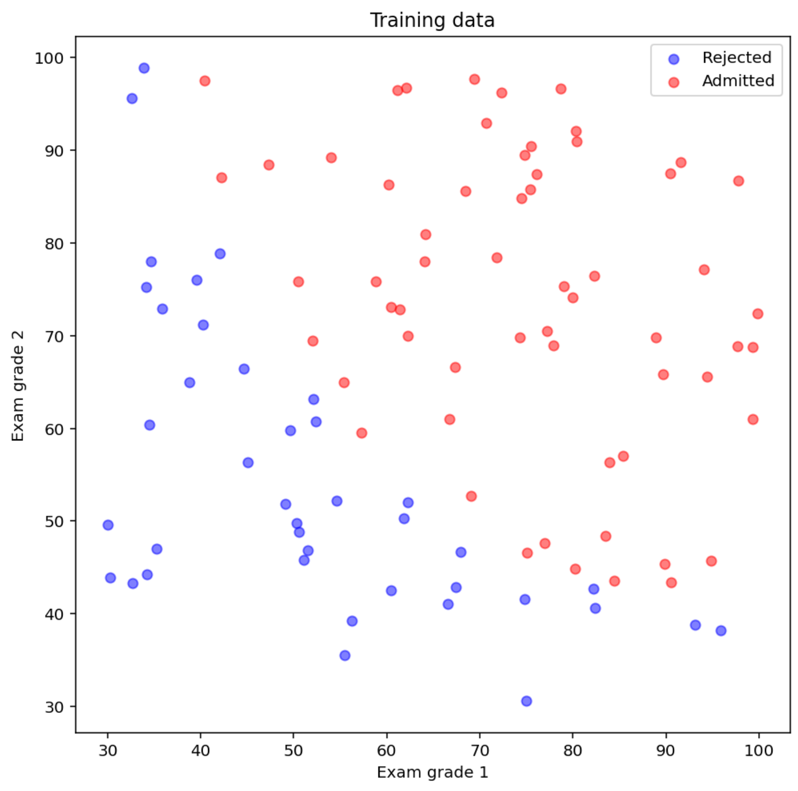
## 2. Explore the dataset distribution

Plot the training data points.

You may use matplotlib function `scatter(x,y)`.

```
x1 = data[:,0].astype(np.float128) # exam grade 1
x2 = data[:,1].astype(np.float128) # exam grade 2
idx = data[:,2].astype(np.float128)
#idx_admit = (idx==1) # index of students who were admitted
#idx_rejec = (idx==0) # index of students who were rejected

x1_idx0    = x1[idx == 0]
x1_idx1    = x1[idx == 1]

x2_idx0    = x2[idx == 0]
x2_idx1    = x2[idx == 1]
```

```
plt.figure(figsize=(8,8))
plt.scatter(x1_idx0, x2_idx0, alpha=0.5, c='b', label='Rejected')
plt.scatter(x1_idx1, x2_idx1, alpha=0.5, c='r', label='Admitted')
plt.title('Training data')
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')
plt.legend()
plt.show()
```



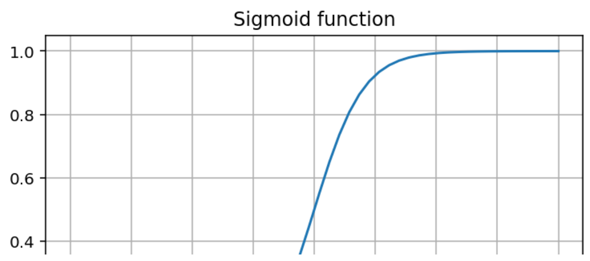
## 3. Sigmoid/logistic function

$$\sigma(\eta) = \frac{1}{1 + \exp^{-\eta}}$$

Define and plot the sigmoid function for values in [-10,10]:

You may use functions `np.exp`, `np.linspace`.

```
def sigmoid(z):
    try:
        return 1 / (1 + np.exp(-z))
    except OverflowError:
        return 1e-9


# plot
x_values = np.linspace(-10,10)

plt.figure(2)
plt.plot(x_values,sigmoid(x_values))
plt.title("Sigmoid function")
plt.grid(True)
```



## 4. Define the prediction function for the classification

The prediction function is defined by:

$$p_w(x) = \sigma(w_0 + w_1 x_{(1)} + w_2 x_{(2)}) = \sigma(w^T x)$$

Implement the prediction function in a vectorised way as follows:

$$X = \begin{bmatrix} 1 & x_{1(1)} & x_{1(2)} \\ 1 & x_{2(1)} & x_{2(2)} \\ \vdots \\ 1 & x_{n(1)} & x_{n(2)} \end{bmatrix} \quad \text{and} \quad w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} \quad \Rightarrow \quad p_w(x) = \sigma(Xw) = \begin{bmatrix} \sigma(w_0 + w_1 x_{1(1)} + w_2 x_{1(2)}) \\ \sigma(w_0 + w_1 x_{2(1)} + w_2 x_{2(2)}) \\ \vdots \\ \sigma(w_0 + w_1 x_{n(1)} + w_2 x_{n(2)}) \end{bmatrix}$$

Use the new function `sigmoid`.

```
# construct the data matrix X
n = len(data)
X = np.array([[1,x1,x2] for x1,x2 in zip(x1, x2)])

# parameters vector
w = np.array([[1.0],[1.0],[1.0]])

# predictive function definition
def f_pred(X,w):

    p = np.dot(X,w)

    return p

y_pred = f_pred(X,w)
```

## 5. Define the classification loss function

Mean Square Error

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} \left(\sigma(w^T x_i) - y_i\right)^2$$

Cross-Entropy

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} \left(-y_i \log(\sigma(w^T x_i)) - (1 - y_i) \log(1 - \sigma(w^T x_i))\right)$$

The vectorized representation is for the mean square error is as follows:

$$L(w) = \frac{1}{n} \Big(p_w(x) - y\Big)^T \Big(p_w(x) - y\Big)$$

The vectorized representation is for the cross-entropy error is as follows:

$$L(w) = \frac{1}{n} \Big( - y^T \log(p_w(x)) - (1 - y)^T \log(1 - p_w(x))\Big)$$

where

$$p_w(x) = \sigma(Xw) = \begin{bmatrix} \sigma(w_0 + w_1 x_{1(1)} + w_2 x_{1(2)}) \\ \sigma(w_0 + w_1 x_{2(1)} + w_2 x_{2(2)}) \\ \vdots \\ \sigma(w_0 + w_1 x_{n(1)} + w_2 x_{n(2)}) \end{bmatrix} \quad \text{and} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

You may use numpy functions `.T` and `np.log`.

```
def mse_loss(label, h_arr):  # mean square error
    loss = (np.dot((h_arr - label).T, (h_arr - label))) / len(h_arr)
    return loss

def ce_loss(label, h_arr):  # cross-entropy error
    loss = -(np.dot(label.T, np.log(h_arr)) + np.dot((1-label).T, np.log(1-h_arr))) / len(h_arr)
    return loss
```

## 6. Define the gradient of the classification loss function

Given the mean square loss

$$L(w) = \frac{1}{n}\Big(p_w(x) - y\Big)^T \Big(p_w(x) - y\Big)$$

The gradient is given by

$$\frac{\partial}{\partial w} L(w) = \frac{2}{n} X^T \Big((p_w(x) - y) \odot (p_w(x) \odot (1 - p_w(x))\Big)$$

Given the cross-entropy loss

$$L(w) = \frac{1}{n}\Big( - y^T \log(p_w(x)) - (1-y)^T \log(1 - p_w(x))\Big)$$

The gradient is given by

$$\frac{\partial}{\partial w} L(w) = \frac{2}{n} X^T (p_w(x) - y)$$

Implement the vectorized version of the gradient of the classification loss function

```
# loss function definition
def loss_logreg(y_pred,y):

    n = len(y)
    #loss = (np.dot((sigmoid(y_pred) - y).T, (sigmoid(y_pred) - y))) / n
    loss = -(np.dot(y.T, np.log(sigmoid(y_pred))) + np.dot((1-y).T, np.log(1-sigmoid(y_pred)))) /
    return loss


# Test loss function
y = data[:,2][:,None] # label
y_pred = f_pred(X,w) # prediction

loss = loss_logreg(y_pred,y)
```

```
# gradient functions
def grad_loss(y_pred, y, X):
    #grad = 2 * np.dot(X.T, np.dot((sigmoid(y_pred)-y), np.dot(sigmoid(y_pred).T, (1-sigmoid(y_pre
    grad = 2 * np.dot(X.T, (sigmoid(y_pred) - y)) / len(y_pred)
    return grad
```

## 7. Implement the gradient descent algorithm

Vectorized implementation for the mean square loss:

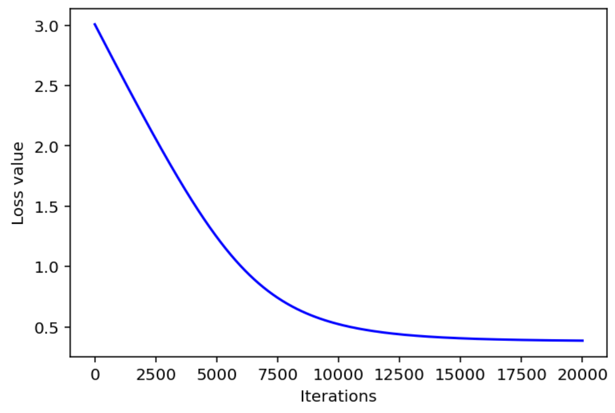
$$w^{k+1} = w^k - \tau \frac{2}{n} X^T \Big( (p_w(x) - y) \odot (p_w(x) \odot (1 - p_w(x)) \Big)$$

Vectorized implementation for the cross-entropy loss:

$$w^{k+1} = w^k - \tau \frac{2}{n} X^T (p_w(x) - y)$$

Plot the loss values $L(w^k)$ w.r.t. iteration $k$ the number of iterations for the both loss functions.

```
# gradient descent function definition
def grad_desc(X, y , w_init=np.array([0,0,0])[:,None] ,tau=1e-4, max_iter=500):

    L_iters = np.zeros([max_iter]) # record the loss values
    w_iters = np.zeros([max_iter,2]) # record the loss values
    w = w_init # initialization

    for i in range(max_iter): # loop over the iterations

        y_pred = f_pred(X,w) # linear predicition function
        grad_f = grad_loss(y_pred,y,X) # gradient of the loss
        w = w - tau* grad_f # update rule of gradient descent
        L_iters[i] = loss_logreg(y_pred,y) # save the current loss value
        w_iters[i,:] = w[0],w[1] # save the current w value

    return w, L_iters, w_iters


# run gradient descent algorithm
start = time.time()
w_init = np.array([-5,0,0])[:,None]
tau = 0.00000005; max_iter = 20000
w, L_iters, w_iters = grad_desc(X,y,w_init,tau,max_iter)


# plot
plt.figure(3)
plt.plot([op for op in range(max_iter)], L_iters, c='blue') # plot the loss curve
plt.xlabel('Iterations')
plt.ylabel('Loss value')
plt.show()

```



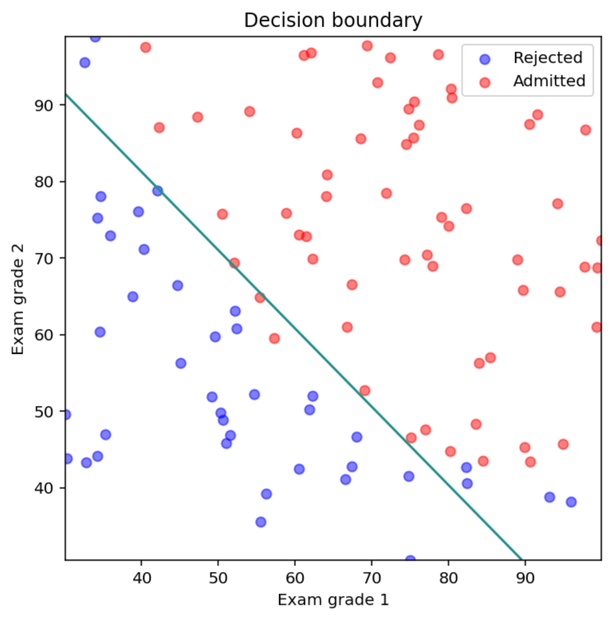
## 8. Plot the decision boundary

The decision boundary is defined by all points

$$x = (x_{(1)}, x_{(2)}) \quad \text{such that} \quad p_w(x) = 0.5$$

You may use numpy and matplotlib functions `np.meshgrid`, `np.linspace`, `reshape`, `contour`.

```
# compute values p(x) for multiple data points x
x1_min, x1_max = X[:,1].min(), X[:,1].max() # min and max of grade 1
x2_min, x2_max = X[:,2].min(), X[:,2].max() # min and max of grade 2
xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max), np.linspace(x2_min, x2_max)) # create meshgri
X2 = np.ones([np.prod(xx1.shape),3])
X2[:,1] = xx1.reshape(-1)
X2[:,2] = xx2.reshape(-1)
p = f_pred(X2,w)
p = p.reshape(50,50)


# plot
plt.figure(4,figsize=(6,6))
plt.scatter(x1_idx0, x2_idx0, alpha=0.5, c='b', label='Rejected')
plt.scatter(x1_idx1, x2_idx1, alpha=0.5, c='r', label='Admitted')
plt.contour(xx1, xx2, p, 0)
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')
plt.legend()
plt.title('Decision boundary')
plt.show()
```



## 9. Comparison with Scikit-learn logistic regression algorithm with the gradient descent with the cross-entropy loss

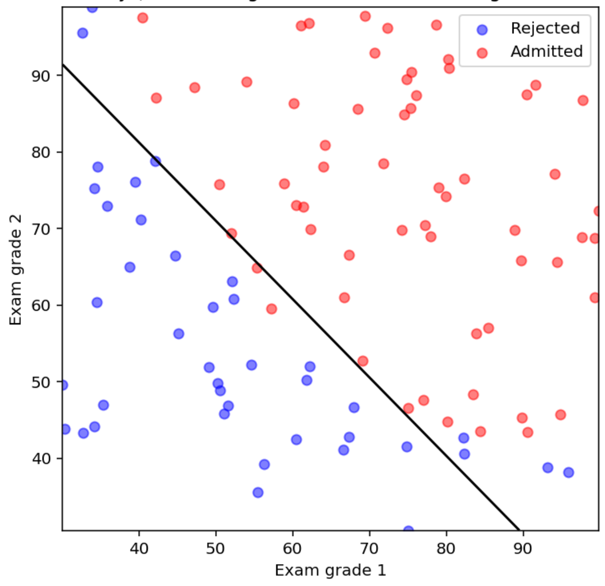
You may use scikit-learn function `LogisticRegression(C=1e6)`.

```
# run logistic regression with scikit-learn
start = time.time()
logreg_sklearn = LogisticRegression(C=1e6, random_state=3)# scikit-learn logistic regression
logreg_sklearn.fit(X, y) # learn the model parameters

# compute loss value
w_sklearn = np.zeros([3,1])
w_sklearn[0,0] = logreg_sklearn.predict(X)[0]
w_sklearn[1:3,0] = logreg_sklearn.predict(X)[1:3]
#loss_sklearn = loss_logreg( )

# plot
```

```
plt.figure(4,figsize=(6,6))
plt.scatter(x1_idx0, x2_idx0, alpha=0.5, c='b', label='Rejected')
plt.scatter(x1_idx1, x2_idx1, alpha=0.5, c='r', label='Admitted')
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')

x1_min, x1_max = X[:,1].min(), X[:,1].max() # grade 1
x2_min, x2_max = X[:,2].min(), X[:,2].max() # grade 2

xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max), np.linspace(x2_min, x2_max)) # create meshgri

X2 = np.ones([np.prod(xx1.shape),3])
X2[:,1] = xx1.reshape(-1)
X2[:,2] = xx2.reshape(-1)

p = f_pred(X2, w)
p = p.reshape(50,50)
p2 = f_pred(X2, w_sklearn)
p2 = p2.reshape(50,50)
plt.contour(xx1, xx2, p, 0, colors='black')
plt.contour(xx1, xx2, p2, 0, colors='magenta')

plt.title('Decision boundary (black with gradient descent and magenta with scikit-learn)')
plt.legend()
plt.show()

```

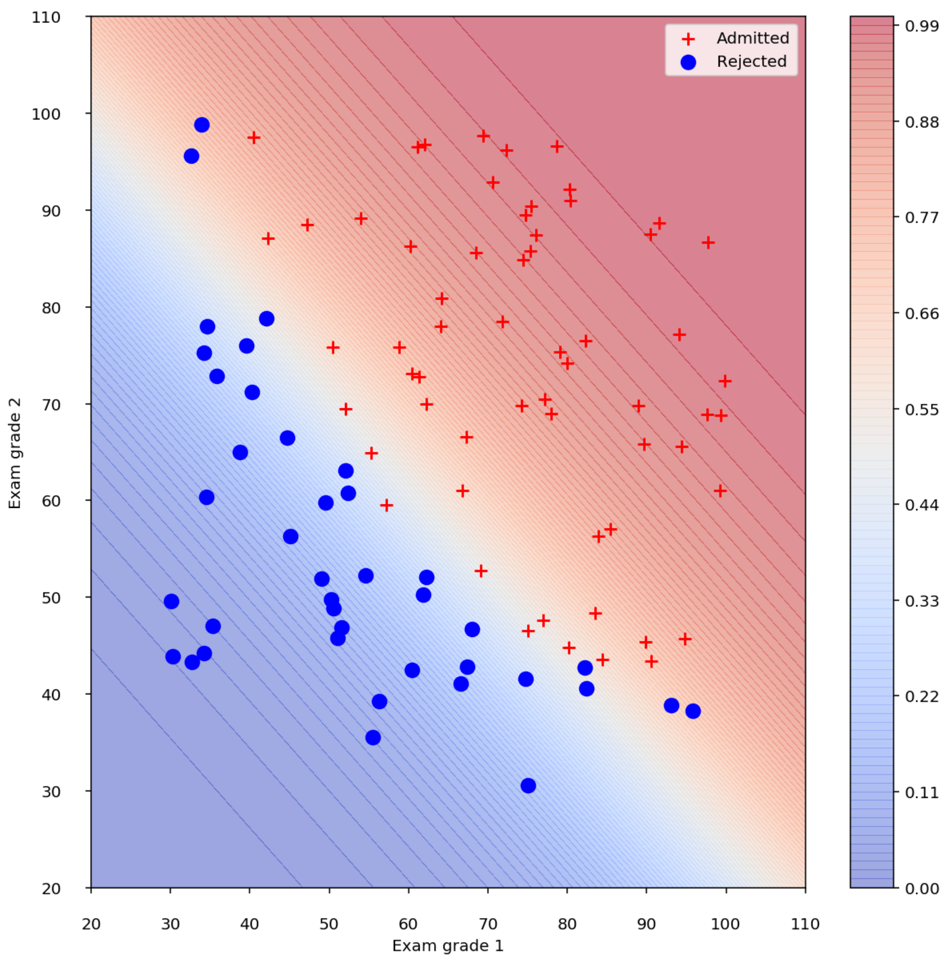
```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:33: UserWarning: No contour levels were found wit
```

Decision boundary (black with gradient descent and magenta with scikit-learn)



## 10. Plot the probability map

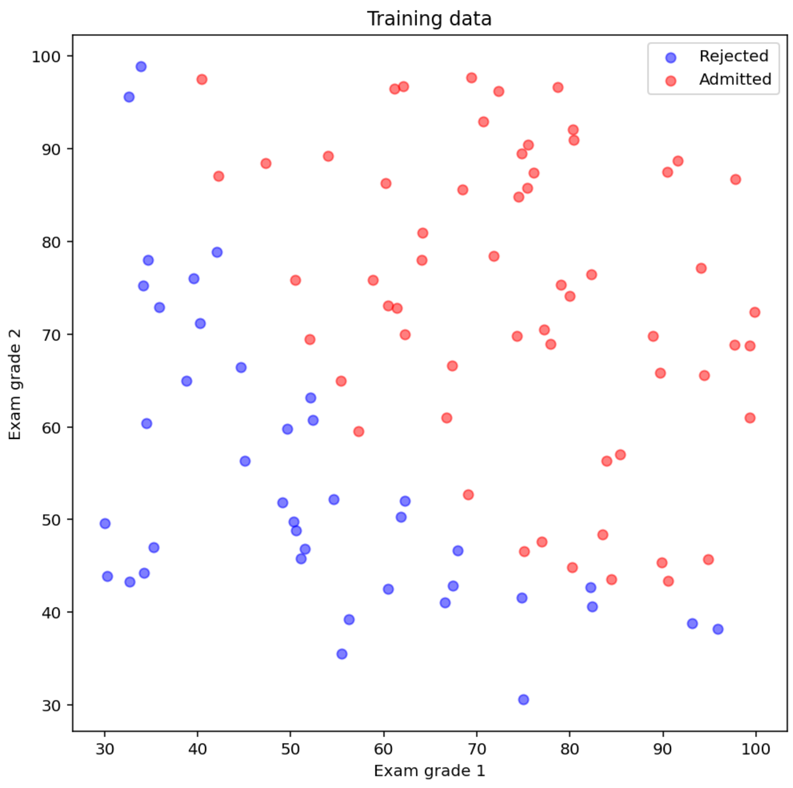
```
num_a = 110
grid_x1 = np.linspace(20,110,num_a)
grid_x2 = np.linspace(20,110,num_a)

score_x1, score_x2 = np.meshgrid( )

Z = np.zeros( )

for i in range(len(score_x1)):
    for j in range(len(score_x2)):
            predict_prob = sigmoid( )

            Z[j, i] = predict_prob

            # actual plotting example
fig = plt.figure(figsize=(10,10))

ax = fig.add_subplot(111)
ax.tick_params( )
ax.set_xlabel('Exam grade 1')
ax.set_ylabel('Exam grade 2')

ax.set_xlim(20, 110)
ax.set_ylim(20, 110)

cf = ax.contourf( )
ax.scatter( )
ax.scatter( )
cbar = fig.colorbar(cf)
cbar.update_ticks()

plt.legend()
plt.show()
```



## Output results

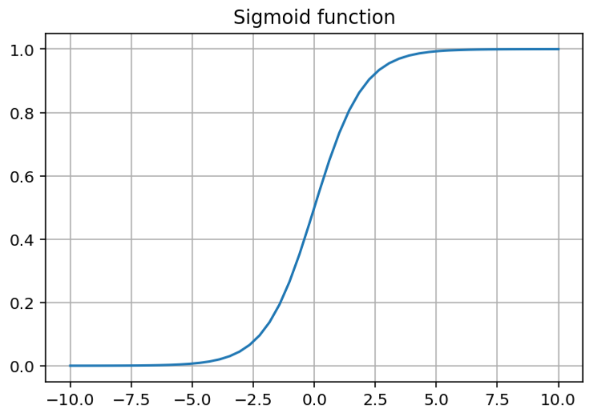
### 1. Plot the dataset in 2D cartesian coordinate system (1pt)

```
plt.figure(figsize=(8,8))
plt.scatter(x1_idx0, x2_idx0, alpha=0.5, c='b', label='Rejected')
plt.scatter(x1_idx1, x2_idx1, alpha=0.5, c='r', label='Admitted')
plt.title('Training data')
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')
plt.legend()
plt.show()
```



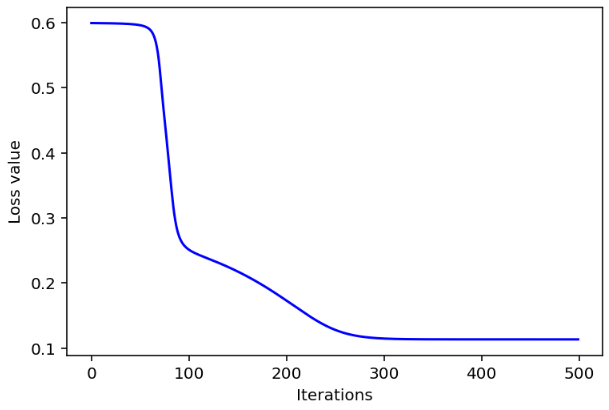
## 2. Plot the sigmoid function (1pt)

```
x_values = np.linspace(-10,10)

plt.figure(2)
plt.plot(x_values,sigmoid(x_values))
plt.title("Sigmoid function")
plt.grid(True)
```



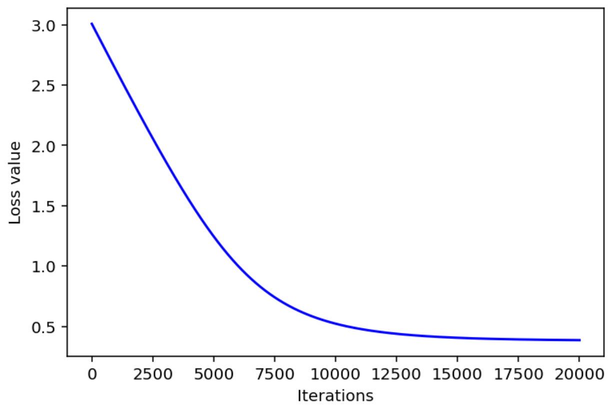
## 3. Plot the loss curve in the course of gradient descent using the mean square error (2pt)

```
# plot
plt.figure(3)
plt.plot([op for op in range(max_iter)], L_iters, c='blue') # plot the loss curve
plt.xlabel('Iterations')
plt.ylabel('Loss value')
plt.show()
```



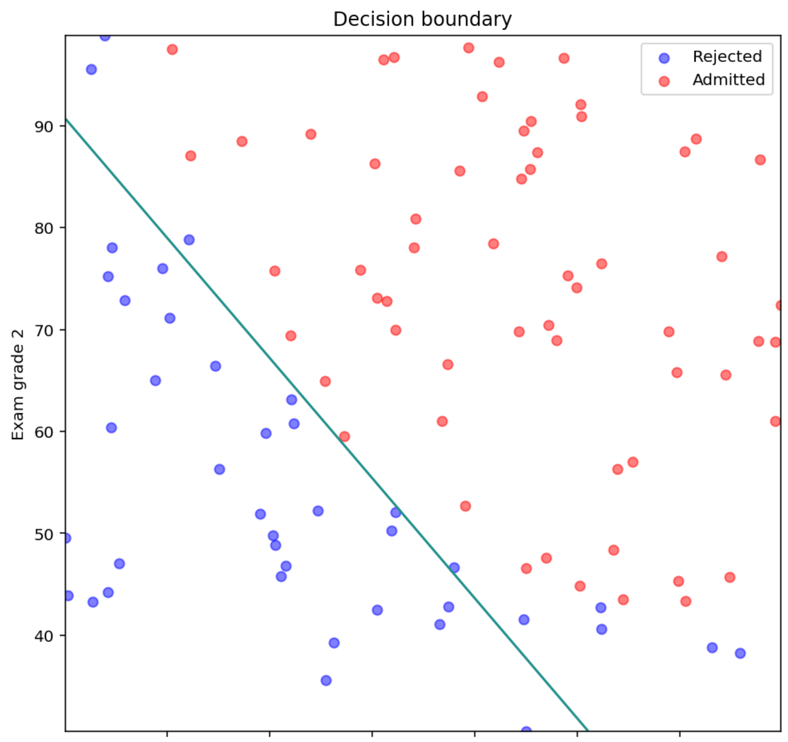
## 4. Plot the loss curve in the course of gradient descent using the cross-entropy error (2pt)

```
# plot
plt.figure(3)
plt.plot([op for op in range(max_iter)], L_iters, c='blue') # plot the loss curve
plt.xlabel('Iterations')
plt.ylabel('Loss value')
plt.show()
```



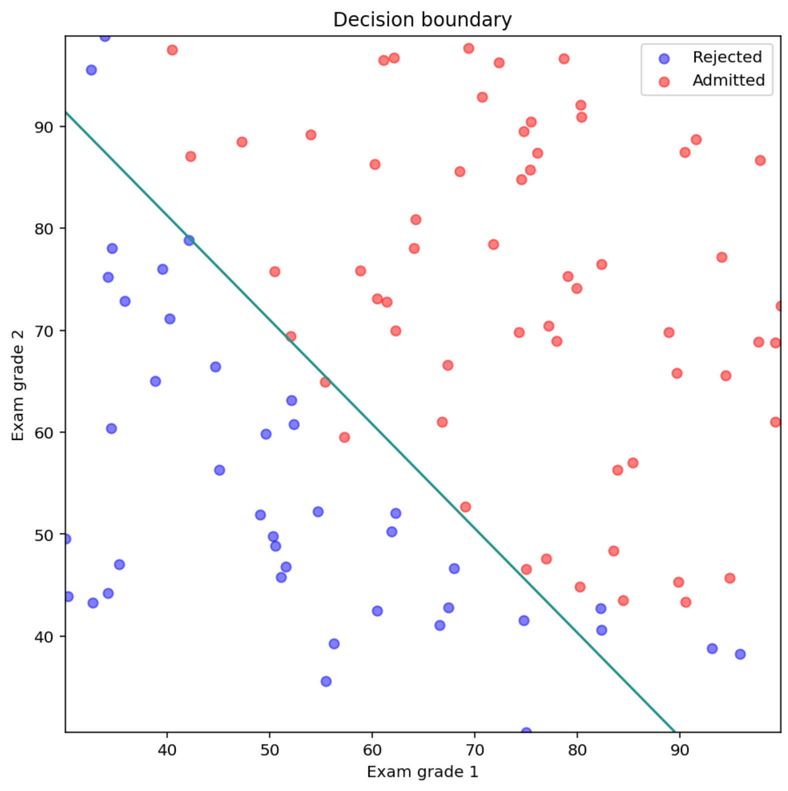
## 5. Plot the decision boundary using the mean square error (2pt)

```
# plot
plt.figure(4,figsize=(8,8))
plt.scatter(x1_idx0, x2_idx0, alpha=0.5, c='b', label='Rejected')
plt.scatter(x1_idx1, x2_idx1, alpha=0.5, c='r', label='Admitted')
plt.contour(xx1, xx2, p, 0)
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')
plt.legend()
plt.title('Decision boundary')
plt.show()
```



## 6. Plot the decision boundary using the cross-entropy error (2pt)

```
# plot
plt.figure(4,figsize=(8,8))
plt.scatter(x1_idx0, x2_idx0, alpha=0.5, c='b', label='Rejected')
plt.scatter(x1_idx1, x2_idx1, alpha=0.5, c='r', label='Admitted')
plt.contour(xx1, xx2, p, 0)
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')
plt.legend()
plt.title('Decision boundary')
plt.show()
```



## 7. Plot the decision boundary using the Scikit-learn logistic regression algorithm (2pt)

## 8. Plot the probability map using the mean square error (2pt)

## 9. Plot the probability map using the cross-entropy error (2pt)