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('/dataset.txt', delimiter=',')

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

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

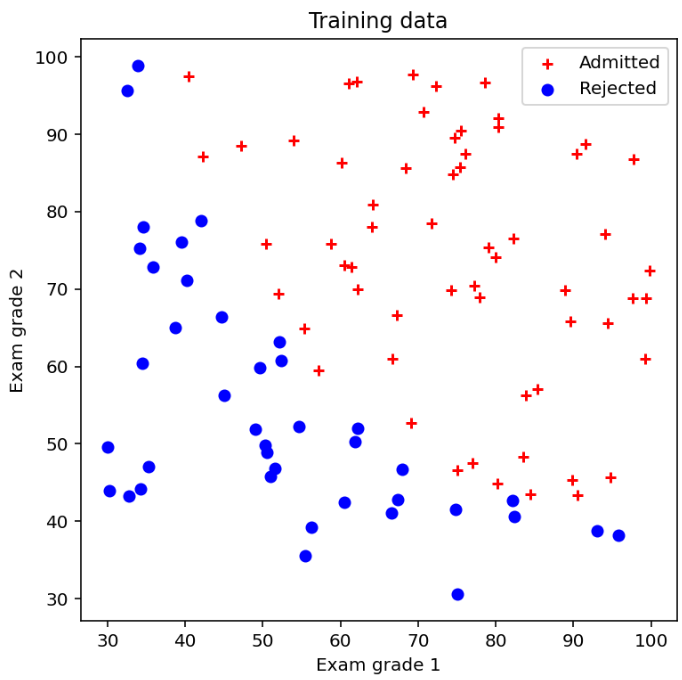
2. Explore the dataset distribution

Plot the training data points.

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

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

fig_1 = plt.figure(figsize = (6,6))
plt.scatter(x1[idx_admit], x2[idx_admit], c= 'r', marker="+", label = 'Admitted')
plt.scatter(x1[idx_rejec], x2[idx_rejec], c= 'b', label = 'Rejected')
plt.title('Training data')
plt.xlabel('Exam grade 1')
```

```
plt.ylabel('Exam grade 2')
plt.legend(loc = 'upper right')
plt.show()
fig_1.savefig('training data.png')
```



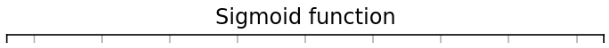
3. Sigmoid/logistic function

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

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

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

```
def sigmoid(z):

    sigmoid_f = 1.0 / (1.0 + np.exp(-z))

    return sigmoid_f


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

fig_2 = plt.figure(2)
plt.plot(x_values,sigmoid(x_values))
plt.title("Sigmoid function")
plt.grid(True)
fig_2.savefig('sigmoid function.png')
```



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 = x1.shape[0]
X = np.ones([n,3])
X[:,1] = x1
X[:,2] = x2

# parameters vector
#w_init = np.array([-10, 0.1, -0.2])[:,None]

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

    p = sigmoid(np.dot(X,w))

    return p

#y_pred = f_pred(X,w)
```

5. Define the classification loss function

Mean Square Error

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

Cross-Entropy

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

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

    return np.mean(np.square((h_arr - label)))

def ce_loss(label, h_arr):  # cross-entropy error

    return np.mean(-label * np.log(h_arr) - (1 - label) * np.log(1 - h_arr))
```

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_mse(y_pred,y):
    n = len(y)
    #  diag(x)y.
    loss = 2 * np.dot(X.T, (y_pred - y) * (y_pred * (1 - y_pred))) / n
    return loss

def loss_ce(y_pred,y):
    n = len(y)
    loss = 2 * np.dot(X.T, (y_pred - y)) / n
    return loss

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

#loss = loss_mse(y_pred,y)
```

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:

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

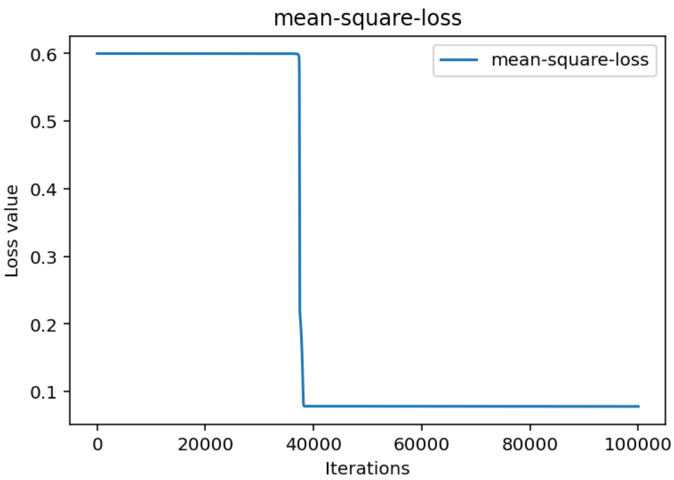
Plot the loss values $L(w^k)$ w.r.t. iteration k the number of iterations for the both loss functions.

```
# gradient descent function definition
def grad_desc_mse(X, y , w_init ,tau=1e-4, max_iter=500):
    L_iters_mse = np.zeros([max_iter])
    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 = loss_mse(y_pred,y) # gradient of the loss
        w = w - tau* grad_f # update rule of gradient descent
        L_iters_mse[i] = mse_loss(y, y_pred) # save the current loss value

        w_iters[i,:] = w[0],w[1] # save the current w value

    return w, L_iters_mse, w_iters
```

```
# gradient descent function definition
def grad_desc_ce(X, y , w_init=np.array([1,1,1])[:,None] ,tau=1e-4, max_iter=500):
    L_iters_ce = 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 = loss_ce(y_pred,y) # gradient of the loss
        w = w - tau* grad_f # update rule of gradient descent
        L_iters_ce[i] = ce_loss(y, y_pred) # save the current loss value

        w_iters[i,:] = w[0],w[1] # save the current w value

    return w, L_iters_ce, w_iters
```

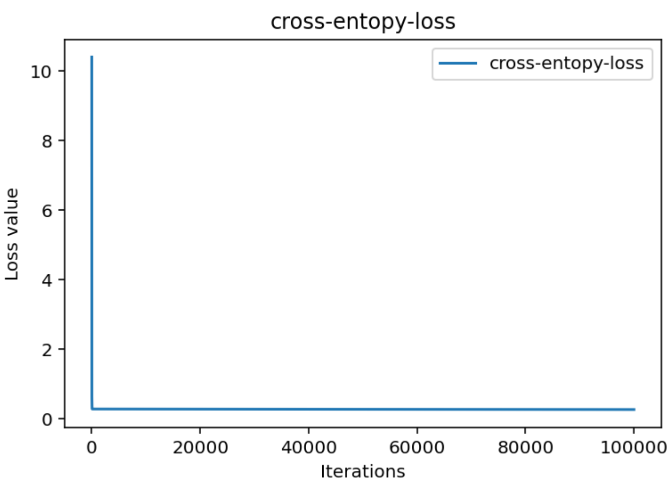
```
# run gradient descent algorithm
start = time.time()
w_init = np.array([-10, 0.1, -0.2])[:,None]
tau = 0.0005; max_iter = 100000

w_mse, L_iters_mse, w_iters_mse = grad_desc_mse(X,y,w_init,tau,max_iter)
# plot
fig_3 = plt.figure(3)
plt.plot(np.linspace(0, max_iter, max_iter), L_iters_mse, label = 'mean-square-loss')
print('mse loss:', L_iters_mse[-1])
print('mse weight:', w_mse)
plt.xlabel('Iterations')
plt.ylabel('Loss value')
plt.legend(loc = 'upper right')
plt.title('mean-square-loss')
plt.show()
fig_3.savefig('mse_loss.png')
```

```
mse loss: 0.07804565596525387
mse weight: [[-10.10936997]
 [  0.08670915]
 [  0.07883981]]
```




```
# run gradient descent algorithm
start = time.time()
w_init = np.array([-10, 0.1, -0.2])[:,None]
tau = 0.0005; max_iter = 100000

w_ce, L_iters_ce, w_iters_ce = grad_desc_ce(X,y,w_init,tau,max_iter)
# plot
fig_4 = plt.figure(3)
plt.plot(np.linspace(0, max_iter, max_iter), L_iters_ce, label = 'cross-entopy-loss')
print('ce loss:', L_iters_ce[-1])
print('ce weight:', w_ce)
plt.xlabel('Iterations')
plt.ylabel('Loss value')
plt.legend(loc = 'upper right')
plt.title('cross-entopy-loss')
plt.show()
fig_4.savefig('ce_loss.png')
```

```
ce loss: 0.25768796845479888
ce weight: [[-11.22782193]
 [  0.09515914]
 [  0.0890087 ]]
```



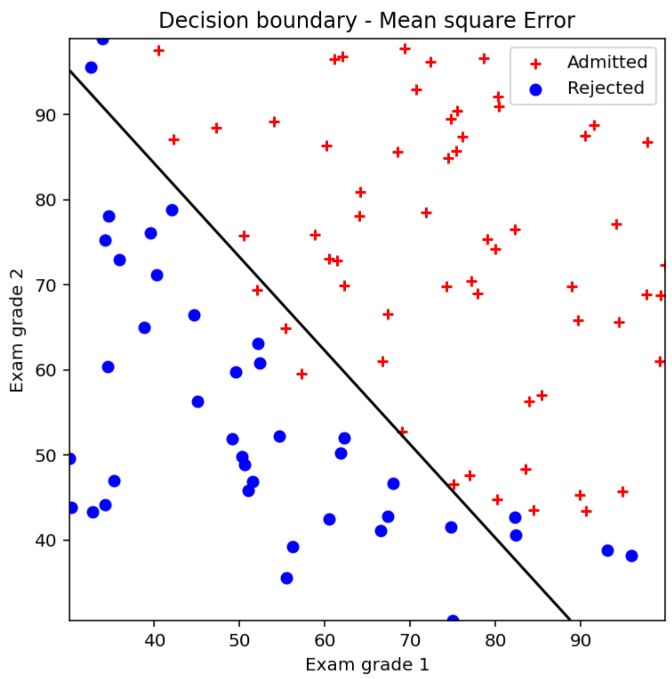
8. Plot the decision boundary

The decision boundary is defined by all points

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

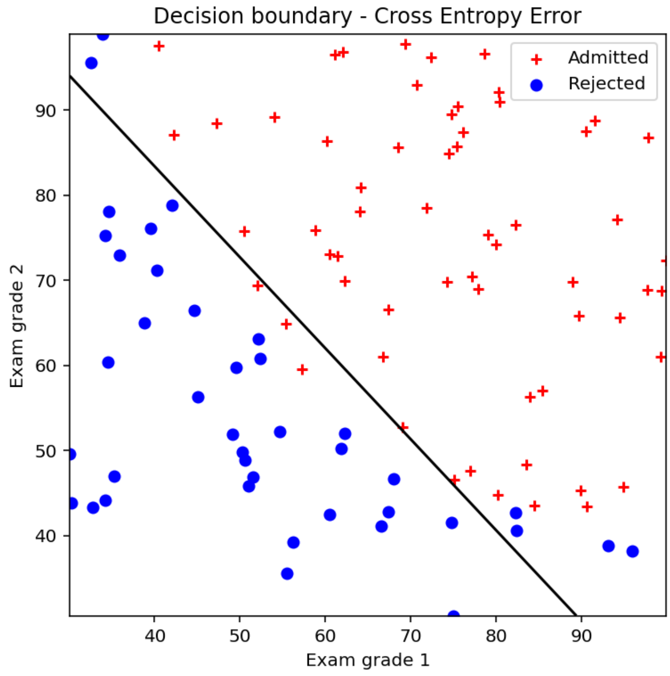
You may use numpy and matplotlib functions `np.meshgrid`, `np.linspace`, `reshape`, `contour`.

```
# 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 meshgrid
X2 = np.ones([np.prod(xx1.shape),3])
X2[:,1] = xx1.reshape(-1)
X2[:,2] = xx2.reshape(-1)
p_mse = f_pred(X2,w_mse)
p_mse = p_mse.reshape(50,50)


# plot
fig_5 = plt.figure(5,figsize=(6,6))
plt.scatter(x1[idx_admit], x2[idx_admit], c= 'r', marker="+", label = 'Admitted')
plt.scatter(x1[idx_rejec], x2[idx_rejec], c= 'b', label = 'Rejected')
plt.contour(xx1, xx2, p_mse, [0.5], colors = 'k')
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')
plt.legend(loc = 'upper right')
plt.title('Decision boundary - Mean square Error')
plt.show()
fig_5.savefig('MSE_boundary.png')
```




```
# 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 meshgrid
X2 = np.ones([np.prod(xx1.shape),3])
X2[:,1] = xx1.reshape(-1)
X2[:,2] = xx2.reshape(-1)
p_ce = f_pred(X2,w_ce)
p_ce = p_ce.reshape(50,50)

# plot
fig_6 = plt.figure(4,figsize=(6,6))
plt.scatter(x1[idx_admit], x2[idx_admit], c= 'r', marker="+", label = 'Admitted')
plt.scatter(x1[idx_rejec], x2[idx_rejec], c= 'b', label = 'Rejected')
plt.contour(xx1, xx2, p_ce, [0.5], colors = 'k')
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')
plt.legend(loc = 'upper right')
plt.title('Decision boundary - Cross Entropy Error')
plt.show()
```

```
fig_6.savefig('CE_boundary.png')
```



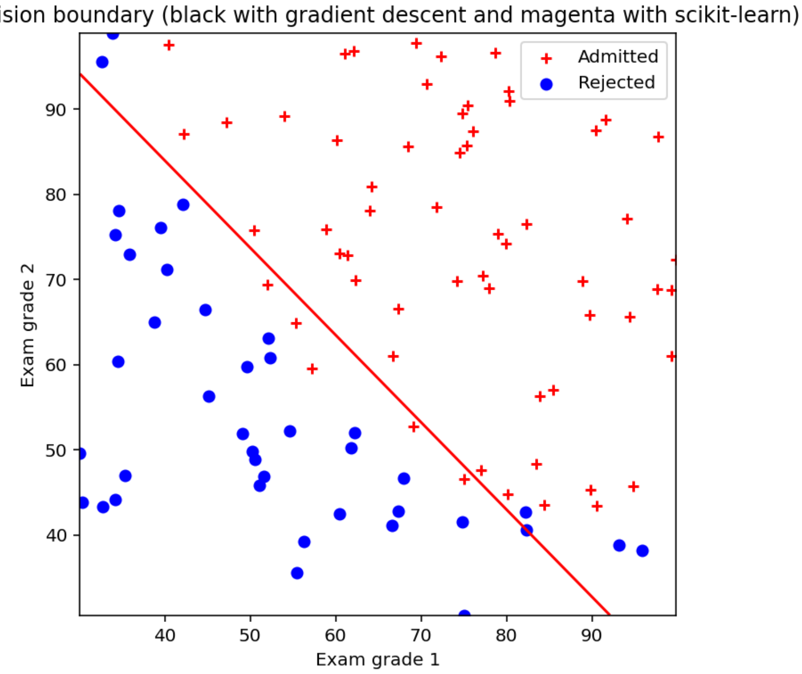
9. Comparison with Scikit-learn logistic regression algorithm with the gradient descent with the cross-entropy loss

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

```
# run logistic regression with scikit-learn
X0 = np.ones([n,2])
X0[:,0] = x1
X0[:,1] = x2

start = time.time()
logreg_sklearn = LogisticRegression(solver = 'lbfgs', tol=0.0005, max_iter=10000) # scikit-learn logistic regression
logreg_sklearn.fit(X0, y.reshape((y.shape[0],))) # learn the model parameters

# compute loss value
print(logreg_sklearn.coef_)
w_sklearn = np.array([1.,1.,1.])[:,None]
w_sklearn[0,0] = logreg_sklearn.intercept_ # bias(w0)
w_sklearn[1:3,0] = logreg_sklearn.coef_[0,0:2].T # w1, w2

loss_sklearn = loss_ce(logreg_sklearn.predict(X0), y.reshape((y.shape[0],)))

# plot
fig_7 = plt.figure(4,figsize=(6,6))
plt.scatter(x1[idx_admit], x2[idx_admit], c= 'r', marker="+", label = 'Admitted')
plt.scatter(x1[idx_rejec], x2[idx_rejec], c= 'b', label = 'Rejected')
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 meshgrid

X3 = np.ones([np.prod(xx1.shape),3])
X3[:,1] = xx1.reshape(-1)
X3[:,2] = xx2.reshape(-1)

p2 = f_pred(X3,w_sklearn)
```

```
p2 = p2.reshape(50,50)
plt.contour(xx1, xx2, p2, [0.5], colors = 'r');

plt.title('Decision boundary (black with gradient descent and magenta with scikit-learn)')
plt.legend(loc = 'upper right')
plt.show()
fig_7.savefig('Decision boundary-scikit-learn.png')
```

```
[[0.20535491 0.2005838 ]]
```

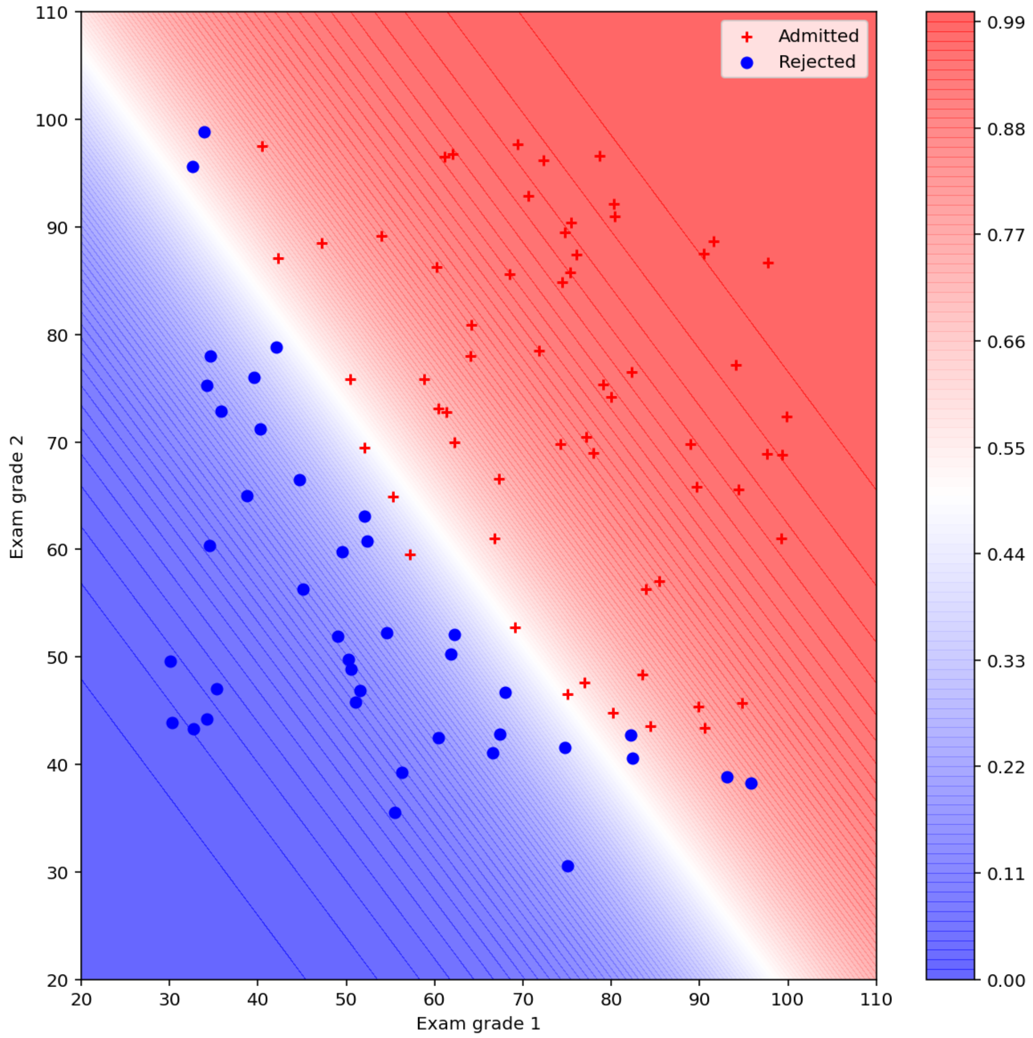


10. Plot the probability map

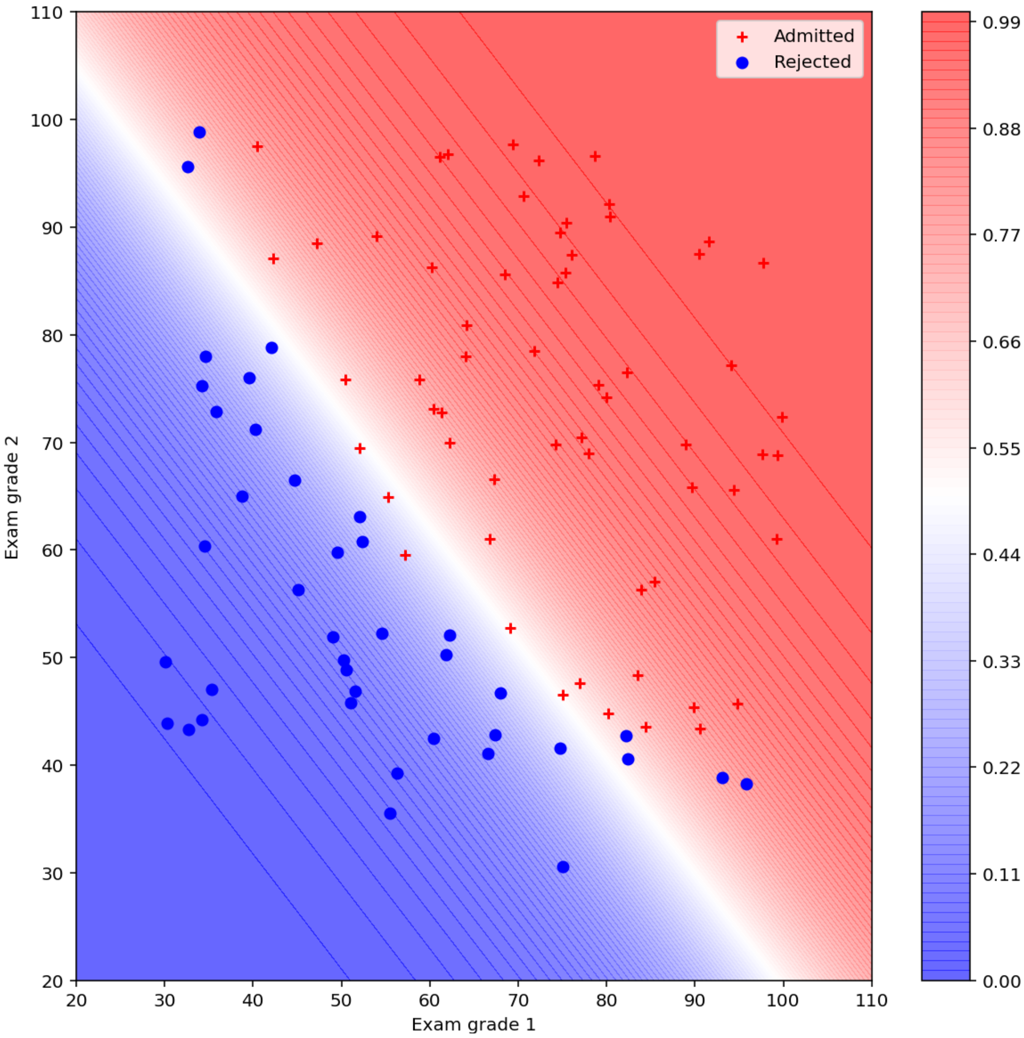
```
num_a = 110
grid_x1 = np.linspace(20,110,num_a)
grid_x2 = np.linspace(20,110,num_a)

score_x1, score_x2 = np.meshgrid(grid_x1, grid_x2)

Z = np.ones([np.prod(score_x1.shape) ,3])
Z[:,1] = score_x1.reshape(-1)
Z[:,2] = score_x2.reshape(-1)
predict_prob_mse = sigmoid(np.dot(Z,w_mse))
predict_prob_mse = predict_prob_mse.reshape(110,110)

predict_prob_ce = sigmoid(np.dot(Z,w_ce))
predict_prob_ce = predict_prob_ce.reshape(110,110)

#for i in range(len(score_x1)):
    #for j in range(len(score_x2)):


            #Z[j, i] = predict_prob

            # actual plotting example
```

```
fig_8 = plt.figure(figsize=(10,10))

ax = fig_8.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_xlim(20, 110)
ax.set_ylim(20, 110)

cf = ax.contourf(score_x1, score_x2, predict_prob_mse, 100, cmap = "bwr", vmin = 0,vmax = 1, alpha = 0.6)
ax.scatter(x1[idx_admit], x2[idx_admit], c= 'r', marker="+", label = 'Admitted')
ax.scatter(x1[idx_rejec], x2[idx_rejec], c= 'b', label = 'Rejected')
cbar = fig_8.colorbar(cf)
cbar.update_ticks()

plt.legend(loc = 'upper right')
plt.show()
fig_8.savefig('Probability Map.png')
```



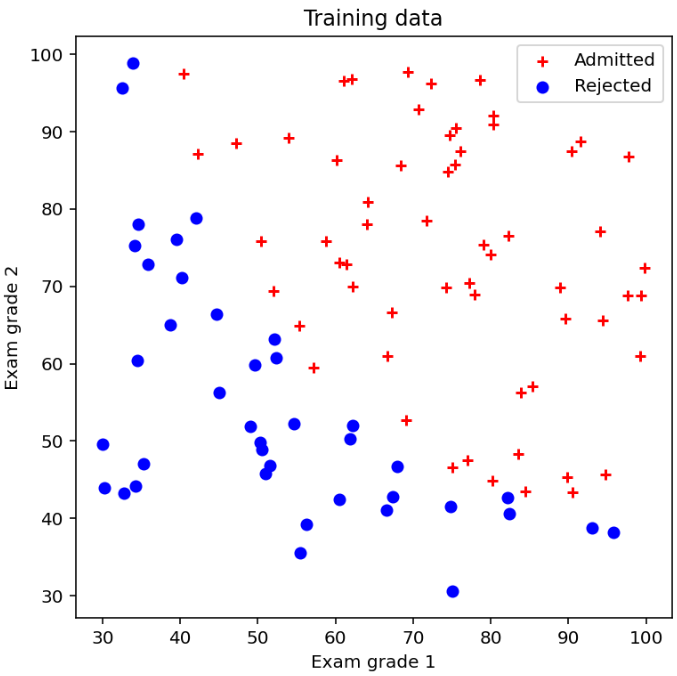

```
fig_9 = plt.figure(figsize=(10,10))

ax = fig_9.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(score_x1, score_x2, predict_prob_ce, 100, cmap = "bwr", vmin = 0,vmax = 1, alpha = 0.6)
ax.scatter(x1[idx_admit], x2[idx_admit], c= 'r', marker="+", label = 'Admitted')
ax.scatter(x1[idx_rejec], x2[idx_rejec], c= 'b', label = 'Rejected')
cbar = fig_9.colorbar(cf)
cbar.update_ticks()

plt.legend(loc = 'upper right')
plt.show()
fig_8.savefig('Probability Map.png')
```



Output results

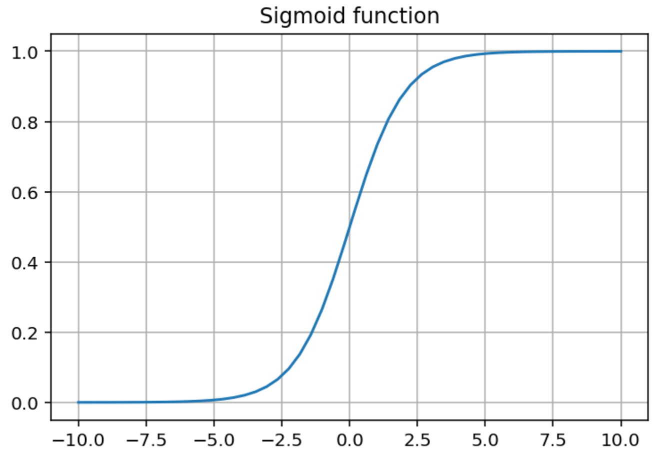
1. Plot the dataset in 2D cartesian coordinate system (1pt)

```
fig_1
```



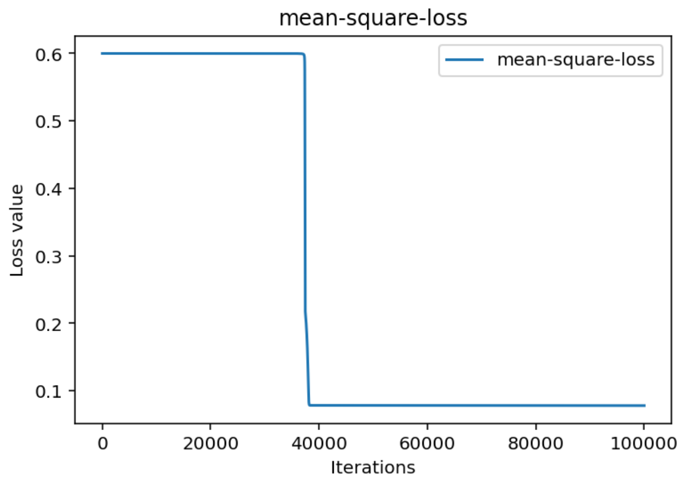
2. Plot the sigmoid function (1pt)

```
1 fig_2
```



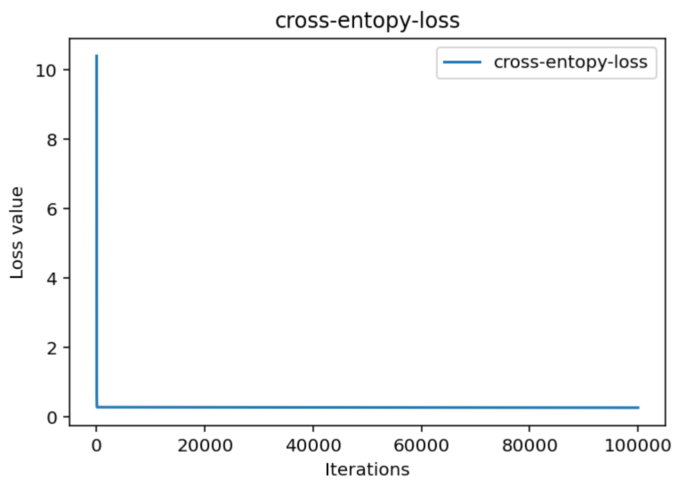
3. Plot the loss curve in the course of gradient descent using the mean square error (2pt)

```
1 fig_3
```



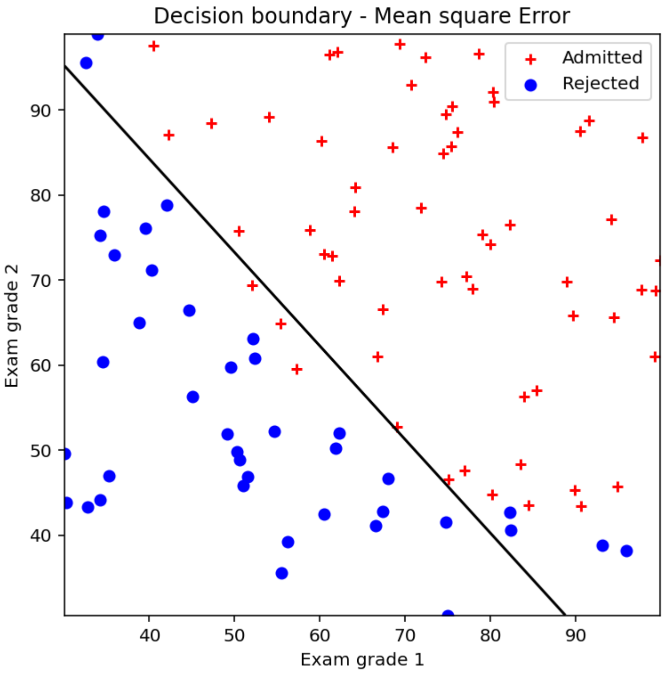
4. Plot the loss curve in the course of gradient descent using the cross-entropy error (2pt)

```
1 fig_4
```



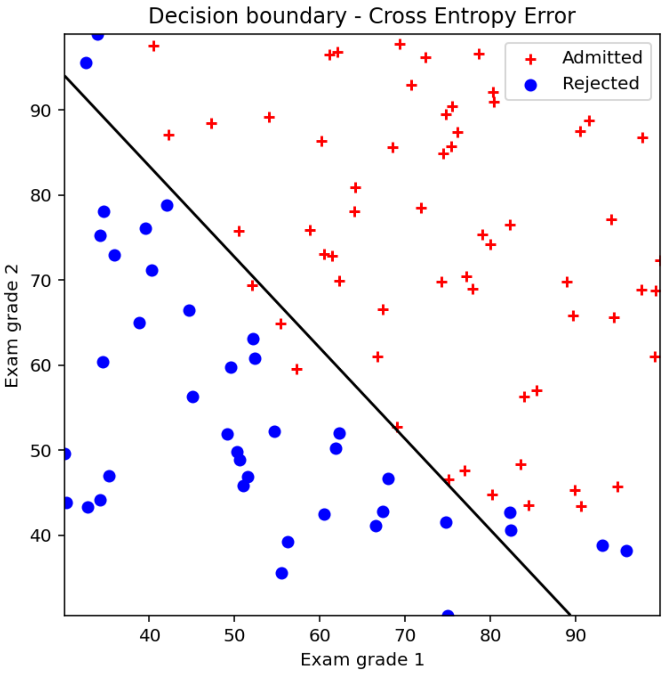
5. Plot the decision boundary using the mean square error (2pt)

```
1    fig_5
```



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

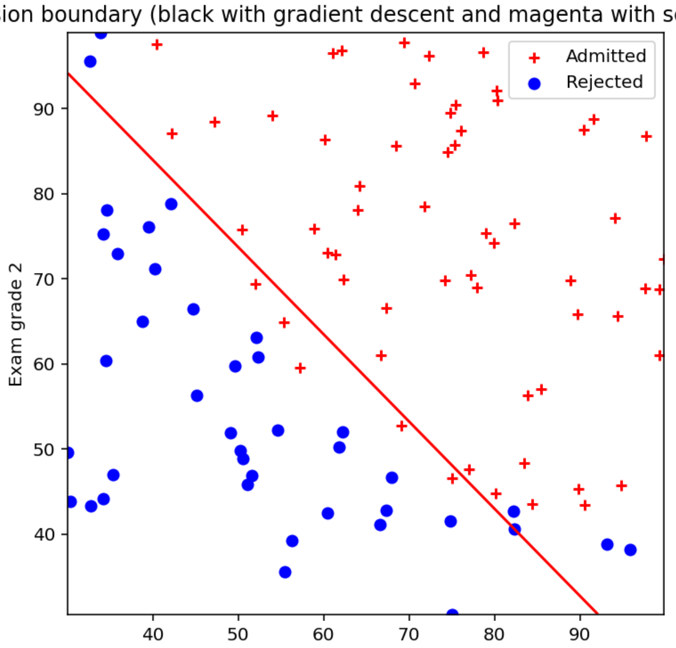
```
1    fig_6
```



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

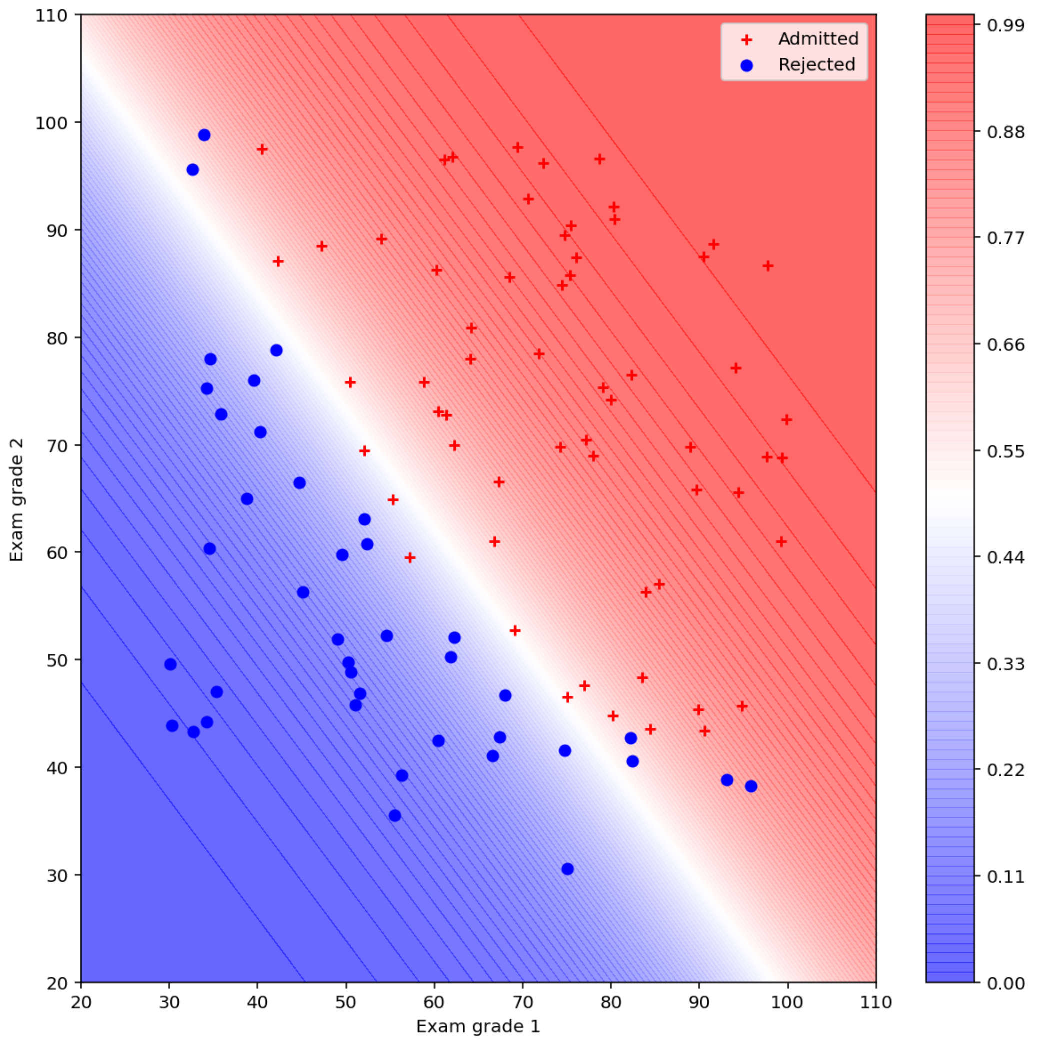
```
1    fig_7
```

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



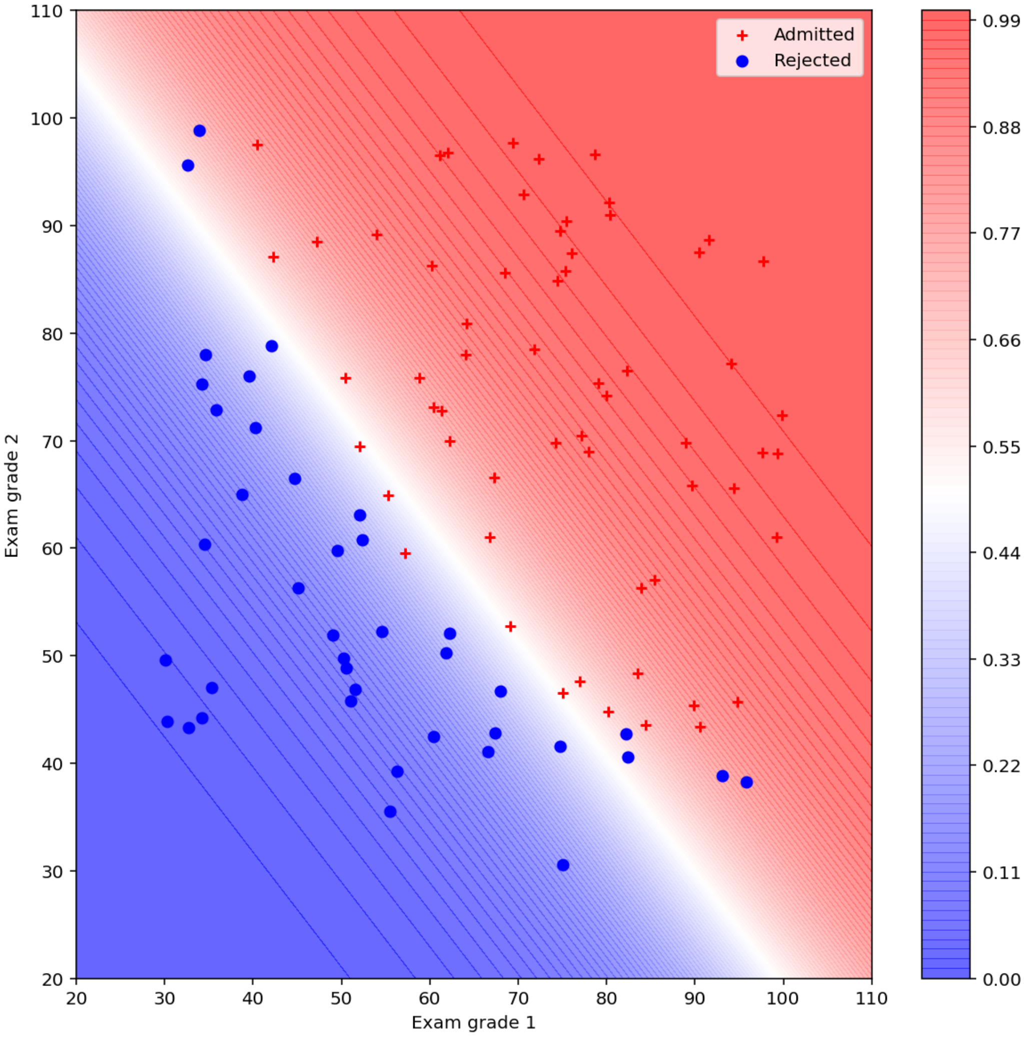
8. Plot the probability map using the mean square error (2pt)

```
1    fig_8
```



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

```
1    fig_9
```



Admitted
Rejected
Exam grade 1
Exam grade 2
20
30
40
50
60
70
80
90
100
110
0.00
0.11
0.22
0.33
0.44
0.55
0.66
0.77
0.88
0.99