Supervised Logistic Regression for Classification

0. Import library

In [143]:

```
# 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
import math
```

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).

In [144]:

```
# 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.

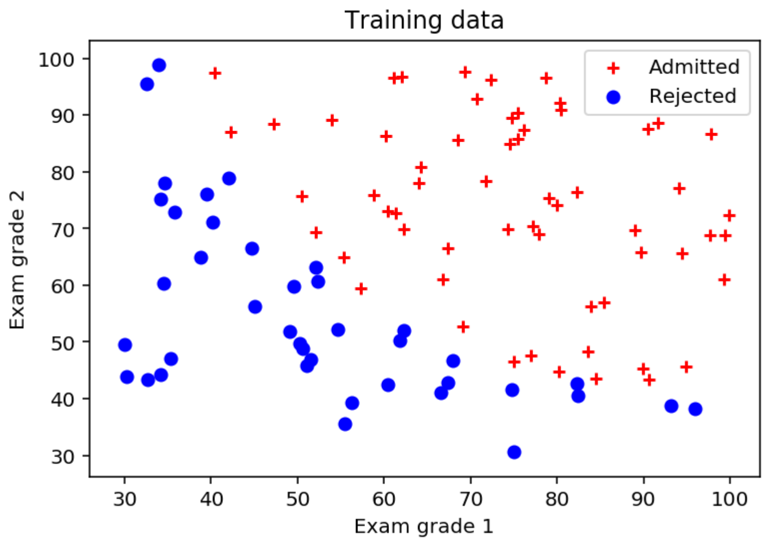
You may use matplotlib function `scatter(x,y)` .

In [145]:

```
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
```

In [146]:

```
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

# 리스트 분배
x1_admit = [x1[i] for i in range(len(x1)) if idx_admit[i]]
x2_admit = [x2[i] for i in range(len(x2)) if idx_admit[i]]

x1_reject = [x1[i] for i in range(len(x1)) if idx_rejec[i]]
x2_reject = [x2[i] for i in range(len(x2)) if idx_rejec[i]]

plt.scatter(x1_admit, x2_admit, c = 'red', label = 'Admitted', marker = '+')
plt.scatter(x1_reject, x2_reject, c = 'blue', label = 'Rejected', marker = 'o')
plt.title('Training data')
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')
plt.legend()
plt.show()
```



3. Sigmoid/logistic function

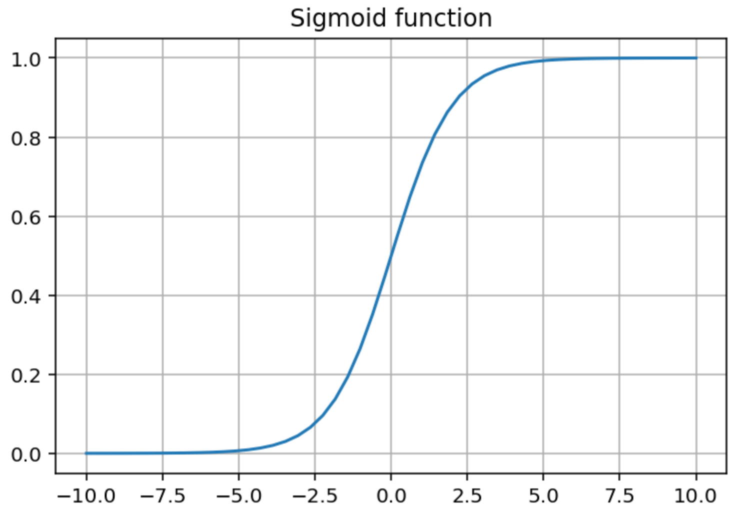
$$ \sigma(\eta) = \frac{1}{1 + \exp^{-\eta}} $$

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

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

In [147]:

```
def sigmoid(z):

    return 1 / (1 + math.e ** -z)


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

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


Sigmoid function
1.0
0.8
0.6
0.4
0.2
0.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

4. Define the prediction function for the classification

The prediction function is defined by:

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

Implement the prediction function in a vectorised way as follows:

$$ X = \left[\begin{array}{cccc} 1 & x_{1(1)} & x_{1(2)} \\ 1 & x_{2(1)} & x_{2(2)} \\ \vdots\\ 1 & x_{n(1)} & x_{n(2)} \end{array} \right] \quad \textrm{ and } \quad w = \left[\begin{array}{cccc} w_0 \\ w_1 \\ w_2 \end{array} \right] \quad \Rightarrow \quad p_w(x) = \sigma(X w) = \left[\begin{array}{cccc} \sigma(w_0 + w_1 x_{1(1)} + w_2 x_{1(2)}) \\ \sigma(w_0 + w_1 x_{2(1)} + w_2 x_{2(2)}) \\ \vdots\\ \sigma(w_0 + w_1 x_{n(1)} + w_2 x_{n(2)}) \end{array} \right] $$

Use the new function `sigmoid`.

In [148]:

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

# parameters vector
w = np.array([0.5, 0, 0])

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

    p = sigmoid(np.dot(X.T, 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

$$ p_w(x)= \sigma(Xw)= \left[\begin{array}{cccc} \sigma(w_0 + w_1 x_{1(1)} + w_2 x_{1(2)}) \\ \sigma(w_0 + w_1 x_{2(1)} + w_2 x_{2(2)}) \\ \vdots\\ \sigma(w_0 + w_1 x_{n(1)} + w_2 x_{n(2)}) \end{array} \right] \quad \textrm{ and } \quad y = \left[\begin{array}{cccc} y_1 \\ y_2 \\ \vdots\\ y_n \end{array} \right] $$

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

In [149]:

```
def mse_loss(label, h_arr):  # mean square error

    return np.mean(np.dot((h_arr - label).T, h_arr - label))

def ce_loss(label, h_arr):  # cross-entropy error
    return np.mean(-((1-label) * np.log(1-h_arr) + label * np.log(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

In [150]:

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

    n = len(y)
    loss = np.sum((y_pred - y) ** 2) / n
    return loss

def grad_mse(y_pred,y, X):

    n = len(y)
    loss = np.dot(X, (y_pred - y) * y_pred * (1 - y_pred)) / n * 2
    return loss

def loss_logreg(y_pred,y):

    n = len(y)
    loss_logreg = np.sum(-((1-y) * np.log(y_pred) + y * np.log(1 -y_pred))) / len(y)

    return loss_logreg

def grad_cross_entropy(y_pred,y, X):

    n = len(y)
    grad_loss = np.dot(X, y_pred - y) / n * 2

    return grad_loss


# def grad_

# Test loss function
y = data[:,2] # label
y_pred = f_pred(X,w) # prediction

loss = loss_logreg(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:

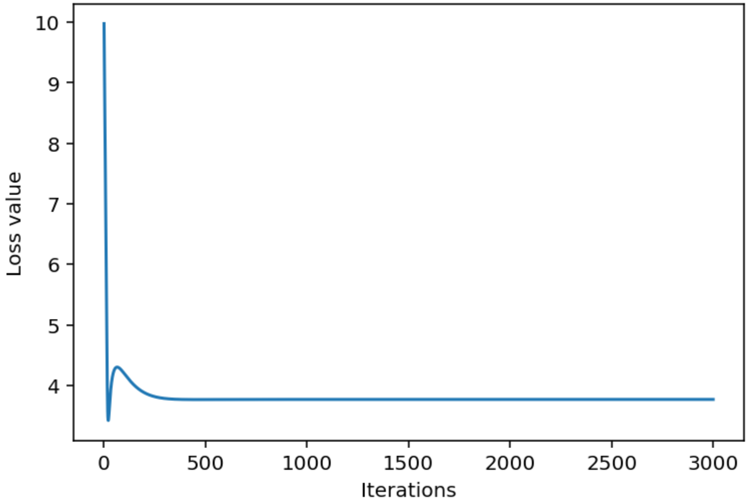
$$ w^{k+1} = w^{k} - \tau \frac{2}{n} X^T(p_w(x)-y) $$

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

In [186]:

```
# gradient descent function definition
def cross_entropy_grad_desc(X, y , w_init = np.array([0,0,0]) ,tau=1e-4, max_iter=500):
     logistic_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_cross_entropy(y_pred, y, X) # gradient of the loss
        w = w - tau* grad_f # update rule of gradient descent
        logistic_iters[i] = loss_logreg(y_pred, y) # save the current loss value
        w_iters[i,:] = w[1:] # save the current w value

    return w, logistic_iters, w_iters

def mse_grad_desc(X, y , w_init = np.array([0,0,0]) ,tau=1e-4, max_iter=500):
    mse_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_mse(y_pred, y, X) # gradient of the loss
        w = w - tau * grad_f # update rule of gradient descent
        mse_iters[i] = loss_mse(y_pred, y) # save the current loss value
        w_iters[i,:] = w[1:] # save the current w value

    return w, mse_iters, w_iters

# run gradient descent algorithm
start = time.time()
w_init = np.array([-20, -0.2, 0.1])
tau = 1e-4; max_iter = 3000

w_cross_entropy, logistic_iters, w_cross_entropy_iters = cross_entropy_grad_desc(X, y, w_init, tau, max_iter)
w_mse, mse_iters,  w_mse_iters = mse_grad_desc(X, y, w_init, tau, max_iter)

# plot
plt.figure(3)
```

```
plt.plot([i for i in range(len(logistic_iters))], logistic_iters)
plt.xlabel('Iterations')
plt.ylabel('Loss value')
plt.show()
```



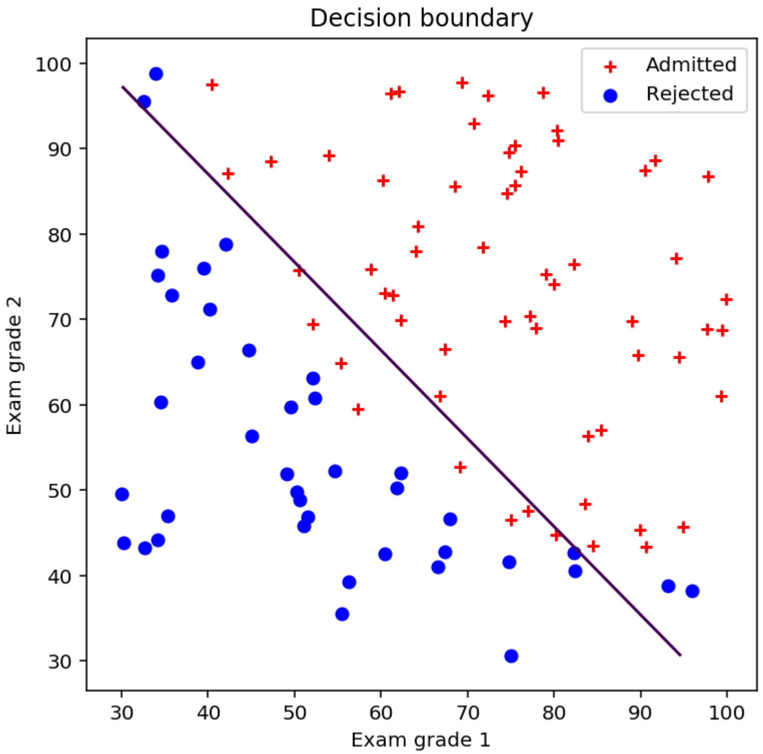
8. Plot the decision boundary

The decision boundary is defined by all points
$$ x=(x_{(1)},x_{(2)}) \quad \textrm{ such that } \quad p_w(x) = 0.5 $$

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

In [187]:

```
# compute values p(x) for multiple data points x
x1_min, x1_max = min(x1), max(x1) # min and max of grade 1
x2_min, x2_max = min(x2), max(x2) # min and max of grade 2
xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max), np.linspace(x2_min, x2_max)) # create meshgrid
Y = w_cross_entropy[0] + w_cross_entropy[1] * xx1 + w_cross_entropy[2] * xx2

# plot
plt.figure(4,figsize=(6,6))
plt.scatter(x1_admit, x2_admit, c = 'red', label = 'Admitted', marker = '+')
plt.scatter(x1_reject, x2_reject, c = 'blue', label = 'Rejected', marker = 'o')
plt.contour(xx1, xx2, Y, [0.5])
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

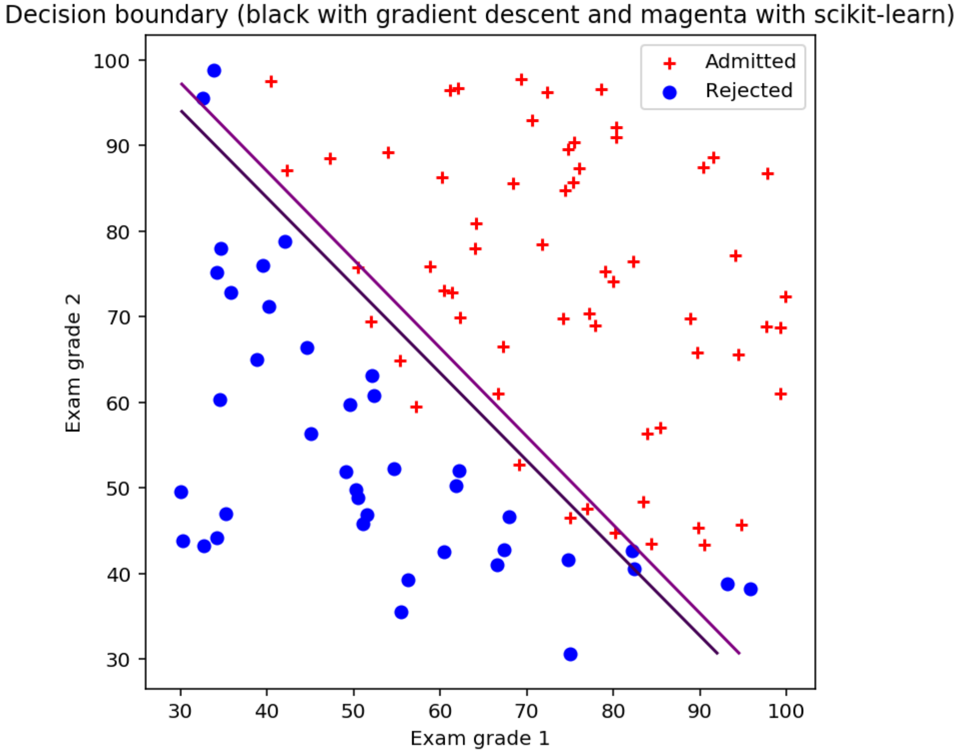
---

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

In [191]:

```
# run logistic regression with scikit-learn
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import *

start = time.time()
logreg_sklearn = LogisticRegression(max_iter=3000, C=1e6) # scikit-learn logistic regression
train_x = data[:,:2]
train_y = data[:,2]
bias = np.ones(100)
train = np.array([bias, train_x[:,0], train_x[:,1]]).T
logreg_sklearn.fit(train_x, y) # learn the model parameters

# compute loss value
w_sklearn = np.zeros([3,1])
w_sklearn[0,0] = logreg_sklearn.intercept_[0]
w_sklearn[1:3,0] = logreg_sklearn.coef_[0]
sklearn_pred = logreg_sklearn.predict(train_x)
loss_sklearn = log_loss(sklearn_pred, y)

x1_min, x1_max = min(x1), max(x1) # min and max of grade 1
x2_min, x2_max = min(x2), max(x2) # min and max of grade 2

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

Y = w_cross_entropy[0] + w_cross_entropy[1] * xx1 + w_cross_entropy[2] * xx2
sklearn_Y = sigmoid(w_sklearn[0][0] + w_sklearn[1][0] * xx1 + w_sklearn[2][0] * xx2)

# plot
plt.figure(4,figsize=(6,6))
plt.scatter(x1_admit, x2_admit, c = 'red', label = 'Admitted', marker = '+')
plt.scatter(x1_reject, x2_reject, c = 'blue', label = 'Rejected', marker = 'o')
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')

plt.contour(xx1, xx2, Y, [0.5], colors = 'purple')
plt.contour(xx1, xx2, sklearn_Y, [0.5])

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

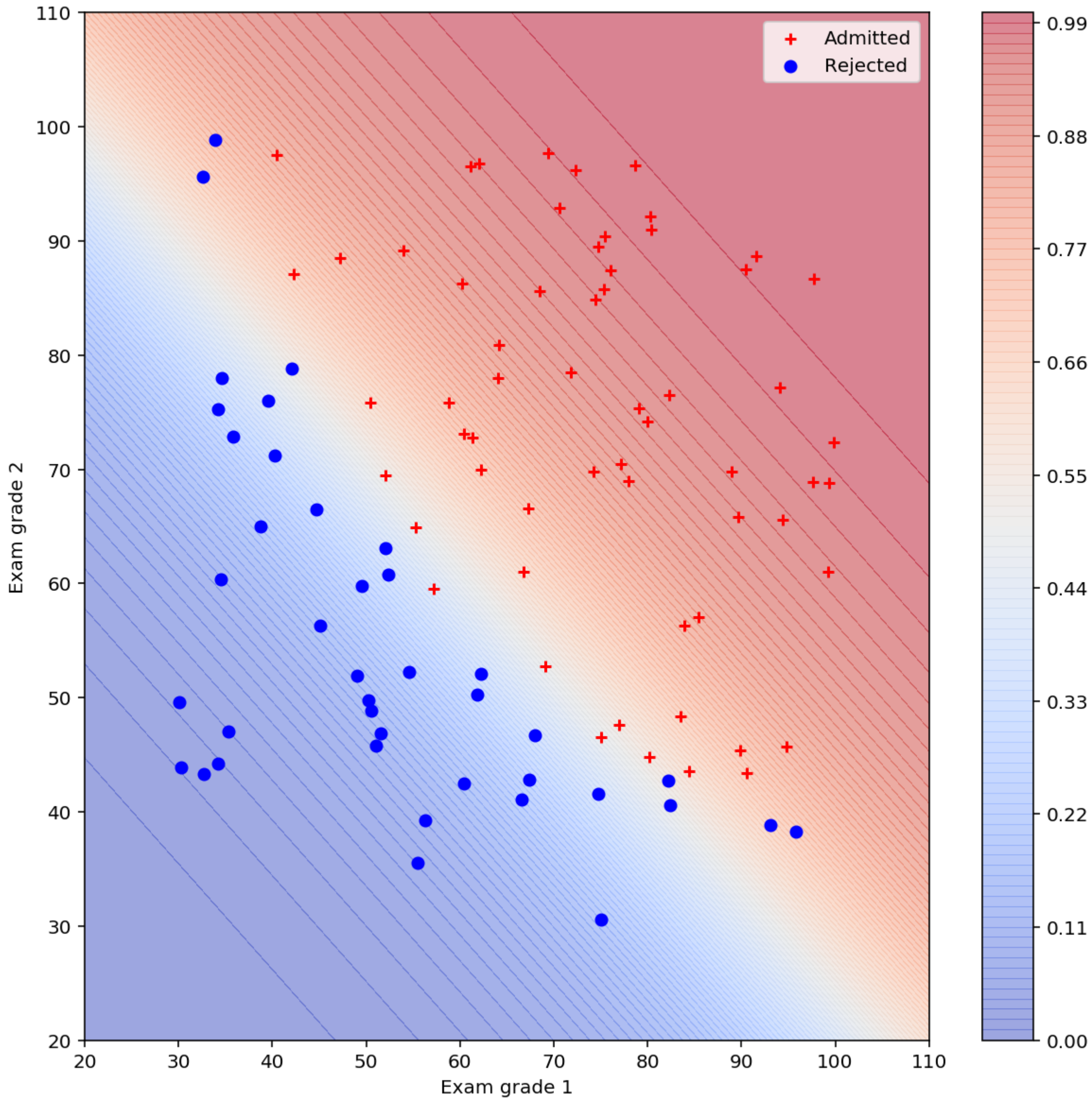
```
plt.legend()
plt.show()
```



10. Plot the probability map

In [184]:

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

Z = np.zeros((len(grid_x1), len(grid_x2)))

for i in range(len(grid_x1)):
    for j in range(len(grid_x2)):
        predict_prob = sigmoid(w_cross_entropy[0] + w_cross_entropy[1] * grid_x1[i] + w_cross_entropy[2] * grid_x2[j])
        Z[j, i] = predict_prob

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

# 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(score_x1, score_x2, np.transpose(Z), cmap=cm.coolwarm, alpha=0.5, levels = np.arange(0, 1.01, 0.01) )
plt.scatter(x1_admit, x2_admit, c = 'red', label = 'Admitted', marker = '+')
plt.scatter(x1_reject, x2_reject, c = 'blue', label = 'Rejected', marker = 'o')

cbar = fig.colorbar(cf)
cbar.update_ticks()

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



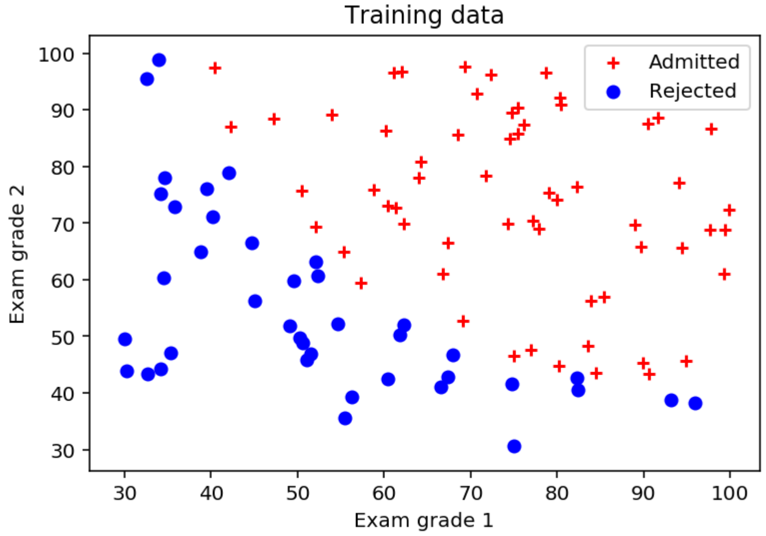
Admitted
Rejected
Exam grade 1
Exam grade 2
20
30
40
50
60
70
80
90
100
110
0.00
0.11
0.22
0.33
0.44
0.55
0.66
0.77
0.88
0.99

Output results

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

In [165]:

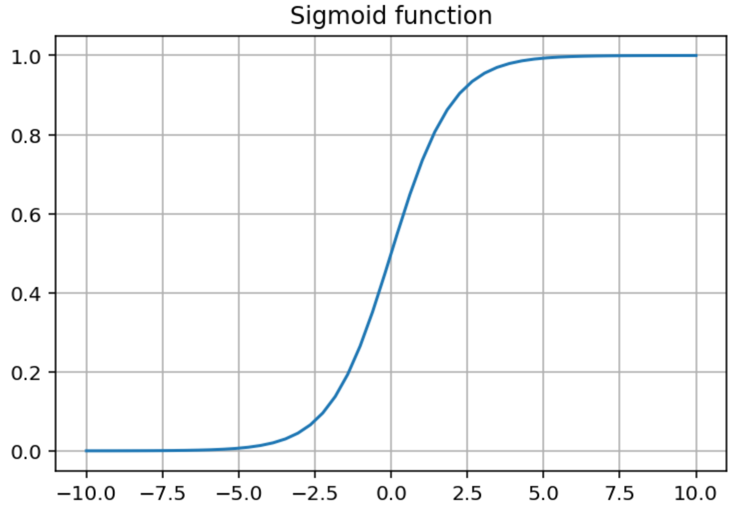
```
plt.scatter(x1_admit, x2_admit, c = 'red', label = 'Admitted', marker = '+')
plt.scatter(x1_reject, x2_reject, c = 'blue', label = 'Rejected', marker = 'o')
plt.title('Training data')
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')
plt.legend()
plt.show()
```



2. Plot the sigmoid function (1pt)

In [166]:

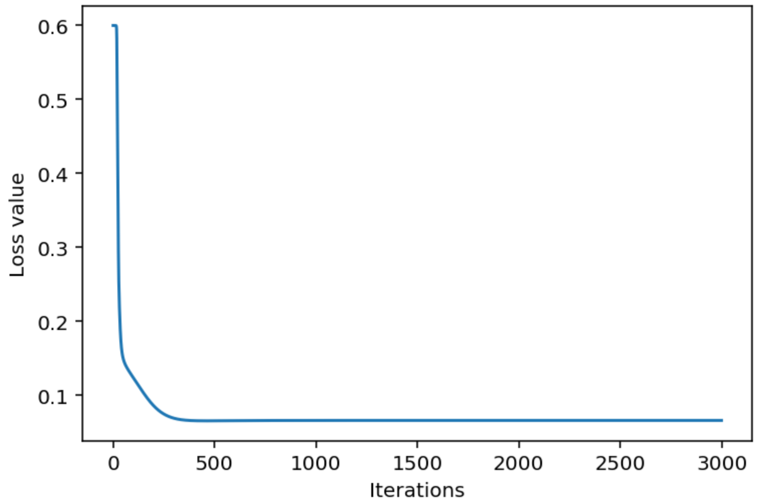
```
plt.figure(2)
plt.plot(x_values,sigmoid(x_values))
plt.title("Sigmoid function")
plt.grid(True)
```



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

In [167]:

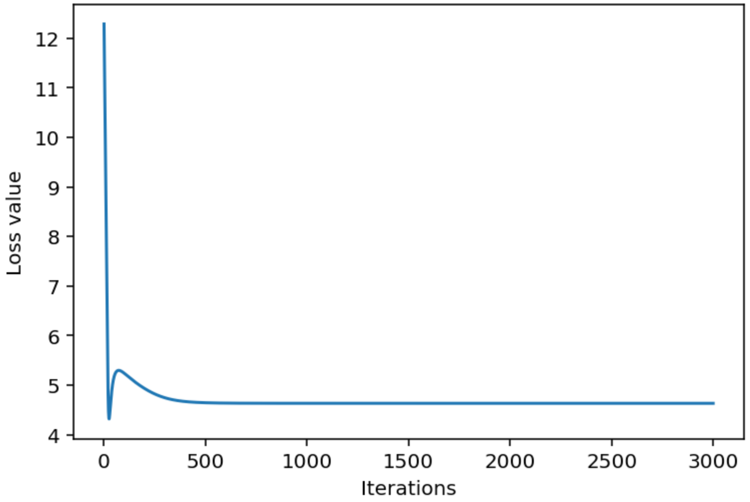
```
# plot
plt.figure(3)
plt.plot([i for i in range(len(mse_iters))], mse_iters)
plt.xlabel('Iterations')
plt.ylabel('Loss value')
plt.show()
```



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

In [168]:

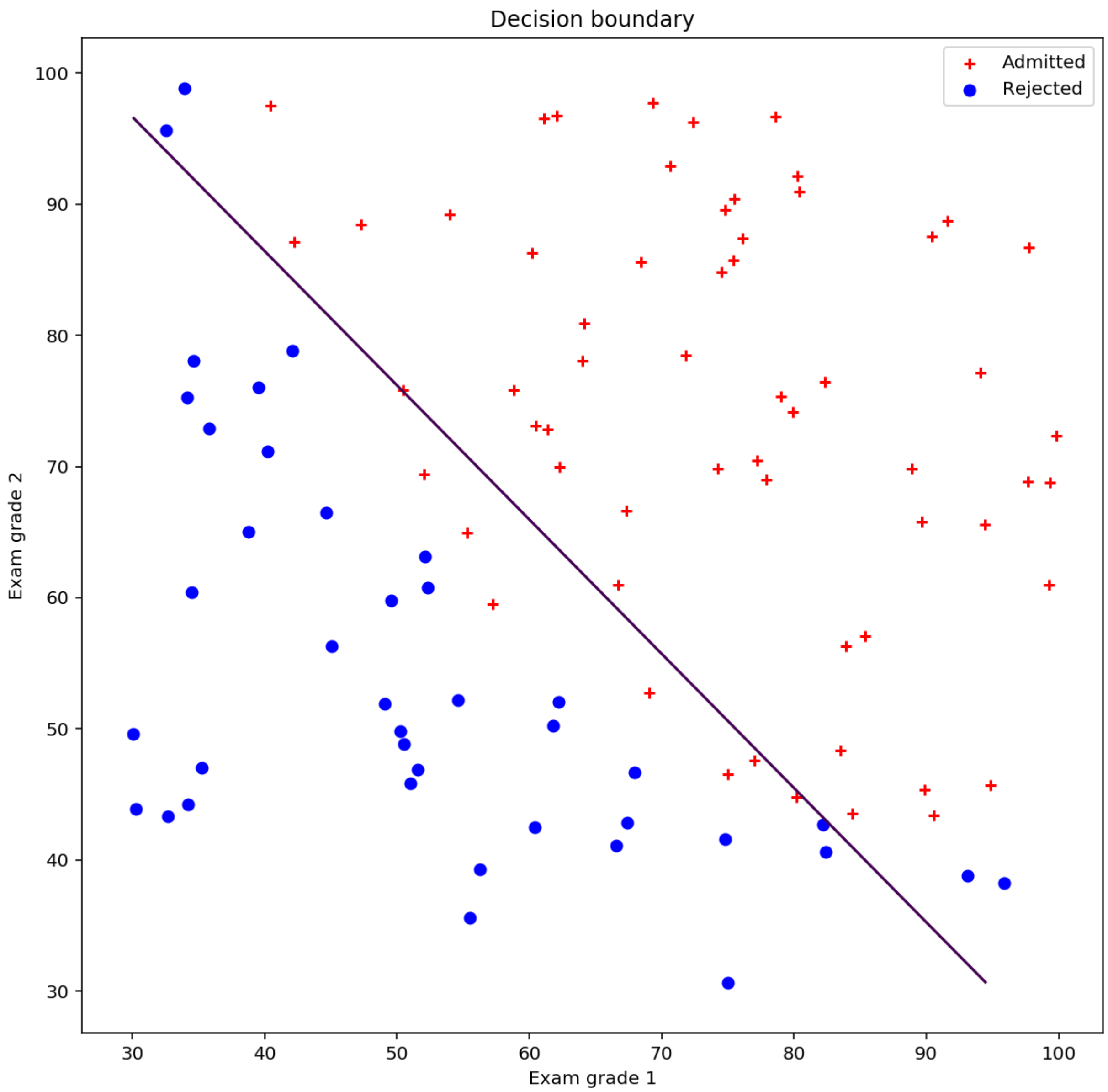
```
# plot
plt.figure(3)
plt.plot([i for i in range(len(logistic_iters))], logistic_iters)
plt.xlabel('Iterations')
plt.ylabel('Loss value')
plt.show()
```



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

In [169]:

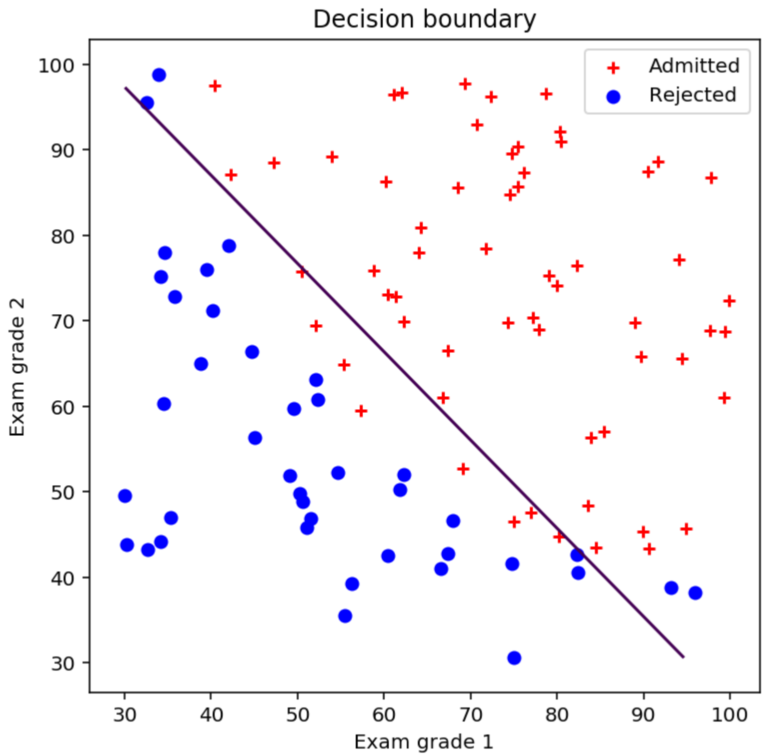
```
# plot
Y = w_mse[0] + w_mse[1] * xx1 + w_mse[2] * xx2

plt.figure(4,figsize=(10,10))
plt.scatter(x1_admit, x2_admit, c = 'red', label = 'Admitted', marker = '+')
plt.scatter(x1_reject, x2_reject, c = 'blue', label = 'Rejected', marker = 'o')
plt.contour(xx1, xx2, Y, [0.5])
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)

In [188]:

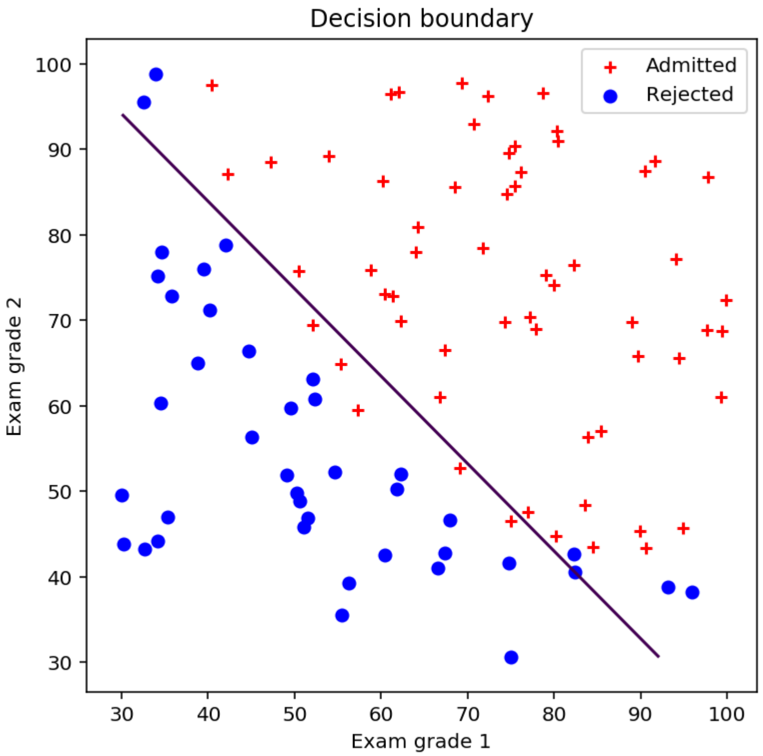
```
Y = w_cross_entropy[0] + w_cross_entropy[1] * xx1 + w_cross_entropy[2] * xx2

# plot
plt.figure(4,figsize=(6,6))
plt.scatter(x1_admit, x2_admit, c = 'red', label = 'Admitted', marker = '+')
plt.scatter(x1_reject, x2_reject, c = 'blue', label = 'Rejected', marker = 'o')
plt.contour(xx1, xx2, Y, [0.5])
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)

In [192]:

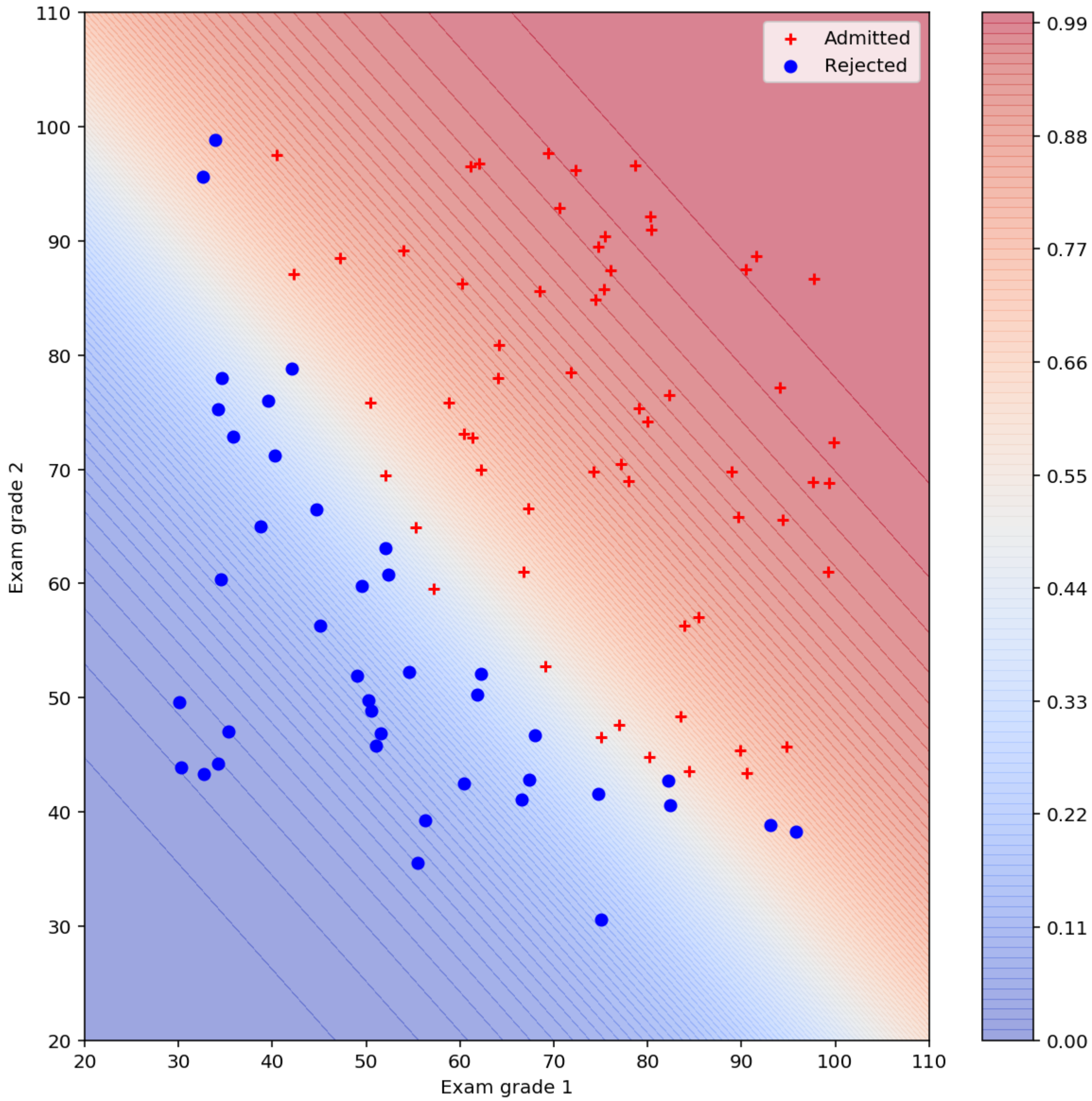
```
# plot
plt.figure(4,figsize=(6,6))
plt.scatter(x1_admit, x2_admit, c = 'red', label = 'Admitted', marker = '+')
plt.scatter(x1_reject, x2_reject, c = 'blue', label = 'Rejected', marker = 'o')
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')

plt.contour(xx1, xx2, sklearn_Y, [0.5])

plt.title('Decision boundary')
plt.legend()
plt.show()
```



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

In [185]:

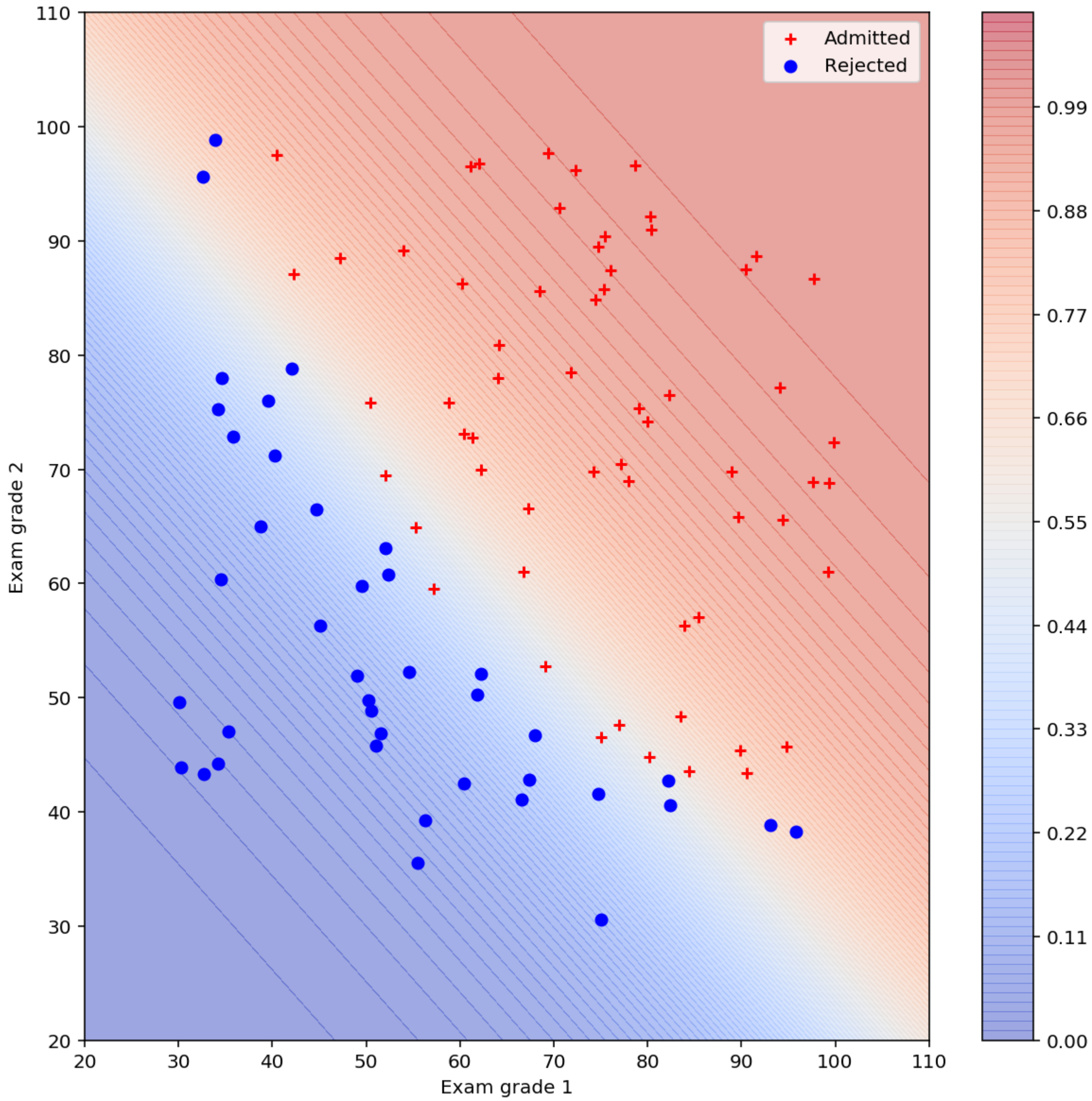
```
num_a = 110
grid_x1 = np.linspace(20,110,num_a)
grid_x2 = np.linspace(20,110,num_a)

Z = np.zeros((len(grid_x1), len(grid_x2)))

for i in range(len(grid_x1)):
    for j in range(len(grid_x2)):
        predict_prob = sigmoid(w_mse[0] + w_mse[1] * grid_x1[i] + w_mse[2] * grid_x2[j])
        Z[j, i] = predict_prob

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

# 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(score_x1, score_x2, np.transpose(Z), cmap=cm.coolwarm, alpha=0.5, levels = np.arange(0, 1.01, 0.01) )
plt.scatter(x1_admit, x2_admit, c = 'red', label = 'Admitted', marker = '+')
plt.scatter(x1_reject, x2_reject, c = 'blue', label = 'Rejected', marker = 'o')

cbar = fig.colorbar(cf)
cbar.update_ticks()

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



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

In [518]:

```
for i in range(len(grid_x1)):
    for j in range(len(grid_x2)):
        predict_prob = sigmoid(w_cross_entropy[0] + w_cross_entropy[1] * grid_x1[i] + w_cross_entropy[2] * grid_x2[j])
        Z[j, i] = predict_prob

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

# 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(score_x1, score_x2, np.transpose(Z), cmap=cm.coolwarm, alpha=0.5, levels = np.arange(0, 1.01, 0.01) )
plt.scatter(x1_admit, x2_admit, c = 'red', label = 'Admitted', marker = '+')
plt.scatter(x1_reject, x2_reject, c = 'blue', label = 'Rejected', marker = 'o')

cbar = fig.colorbar(cf)
cbar.update_ticks()

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



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