Supervised Logistic Regression for Classification

0. Import library

In [1]:

```
# 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).

In [2]:

```
# 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
```

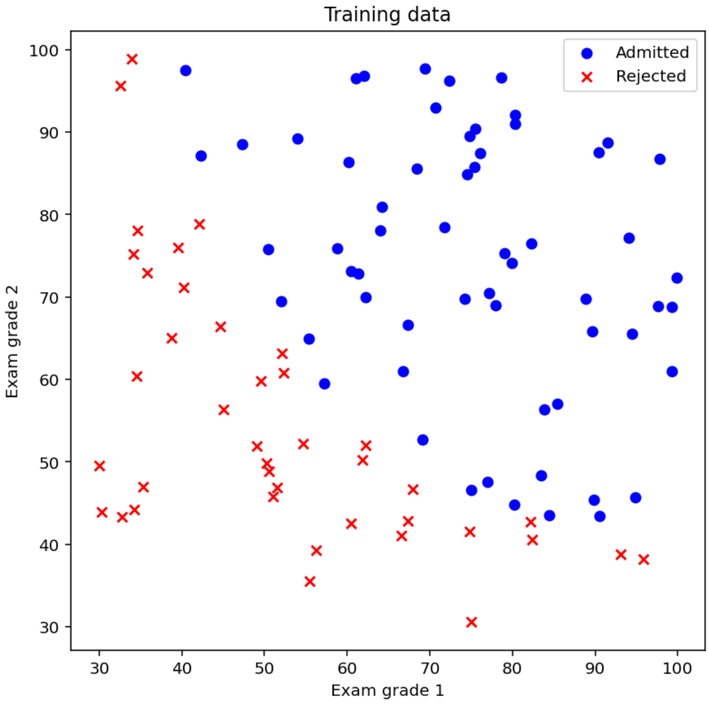
2. Explore the dataset distribution

Plot the training data points.

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

In [16]:

```
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

plt.figure(figsize = (7, 7))
plt.scatter(x1[idx_admit], x2[idx_admit], c = 'b', marker = 'o', label = 'Admitted')
plt.scatter(x1[idx_rejec], x2[idx_rejec], c = 'r', marker = 'x', label = 'Rejected' )
plt.title('Training data')
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')
plt.legend(loc = 'upper right')
plt.show()
```



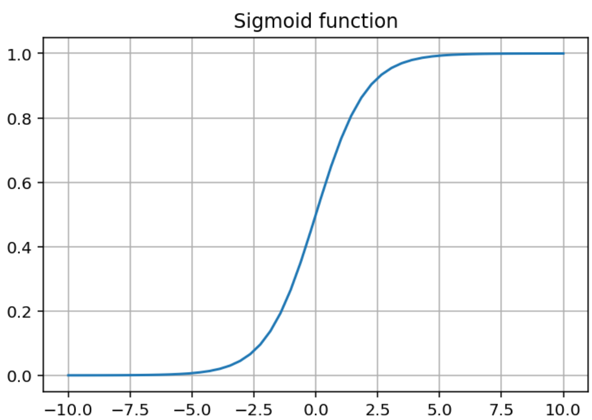
3. Sigmoid/logistic function

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

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

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

In [18]:
```
def sigmoid(z):

    sigmoid_f = 1 / (1 + np.exp(-z))

    return sigmoid_f


# 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:

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

$$p_w(x) = \sigma(Xw) = \begin{bmatrix} \sigma(w_0 + w_1 x_{1(1)} + w_2 x_{1(2)}) \\ \sigma(w_0 + w_1 x_{2(1)} + w_2 x_{2(2)}) \\ \vdots \\ \sigma(w_0 + w_1 x_{n(1)} + w_2 x_{n(2)}) \end{bmatrix}$$

Use the new function `sigmoid` .

In [516]:
```
# construct the data matrix X
n = x1.size
X = np.insert(np.append(np.array(x1).reshape(-1, 1),
              np.array(x2).reshape(-1, 1), axis = 1), 0, 1, axis = 1)

# parameters vector
w = np.array([[-5], [0.1], [0]])

# 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

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

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` .

In [405]:
```
# label = predict output
# h_arr = 정답 label
def mse_loss(label, h_arr):  # mean square error
```

```python
    temp = label - h_arr
    return np.mean(np.dot(temp.T, temp))

def ce_loss(label, h_arr):  # cross-entropy error
    epsilon = 1e-5
    temp = np.dot(-(h_arr.T), np.log(label + epsilon)) - np.dot((1 - h_arr).T, np.log(1 - label + epsilon))
    return np.mean(temp)
```

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 [406]:
```python
# loss function of cross-entropy
def grad_loss_mse(y_pred,y):

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

    return loss

# loss function of cross-entropy
def grad_loss_ce(y_pred,y):

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

    return loss


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

gloss_mse = grad_loss_mse(y_pred,y)
gloss_ce = grad_loss_ce(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:

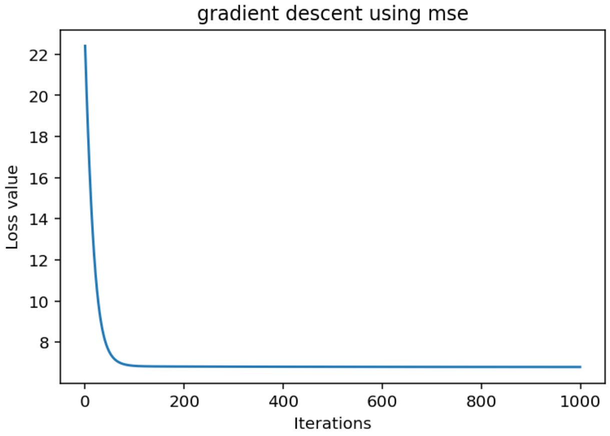
$$w^{k+1} = w^k - \tau\frac{2}{n}X^T(p_w(x) - y)$$

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

In [407]:

```python
# gradient descent function definition
def grad_desc_mse(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_mse(y_pred,y) # gradient of the loss
        w = w - tau* grad_f # update rule of gradient descent
        L_iters[i] = mse_loss(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([-15, 0.1, 0.1])[:,None]
tau = 0.0001; max_iter = 1000
w_mse, L_iters, w_iters = grad_desc_mse(X,y,w_init,tau,max_iter)

print(w_mse)
print(L_iters[max_iter-1])


# plot
plt.figure(3)
plt.plot(L_iters)
plt.title('gradient descent using mse')
plt.xlabel('Iterations')
plt.ylabel('Loss value')
plt.show()
```

```
[[-14.99976342]
 [  0.1266953 ]
 [  0.11843229]]
6.786704458288824
```

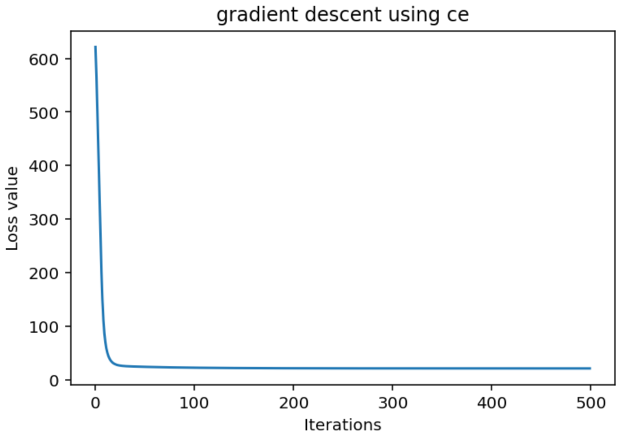


In [418]:

```python
# gradient descent function definition
def grad_desc_ce(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_ce(y_pred,y) # gradient of the loss
        w = w - tau* grad_f # update rule of gradient descent
        L_iters[i] = ce_loss(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([-20, 0.1, 0.02])[:,None]
```

```
tau = 0.0001; max_iter = 500
w_ce, L_iters, w_iters = grad_desc_ce(X,y,w_init,tau,max_iter)

print(w_ce)
print(L_iters[max_iter-1])

# plot
plt.figure(3)
plt.plot(L_iters)
plt.title('gradient descent using ce')
plt.xlabel('Iterations')
plt.ylabel('Loss value')
plt.show()
```

```
[[-19.99878008]
 [  0.16564094]
 [  0.15900519]]
20.830791631536393
```



8. Plot the decision boundary

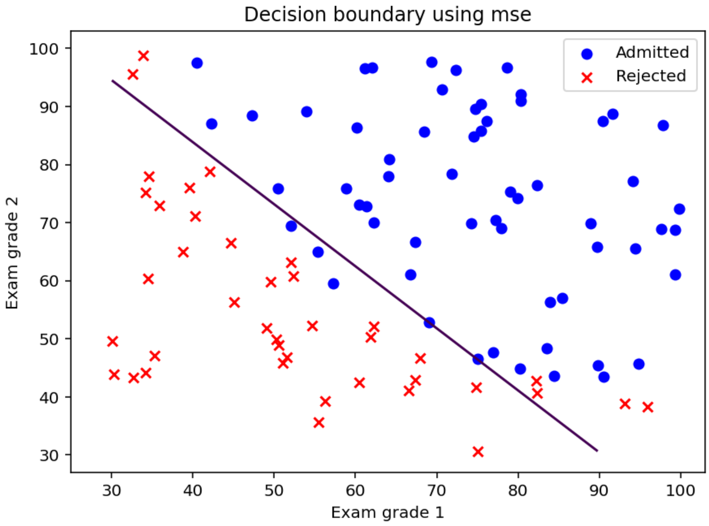
The decision boundary is defined by all points

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

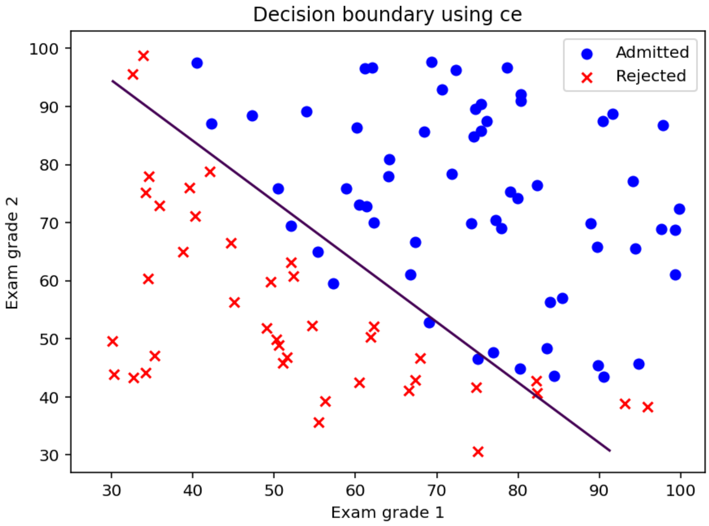
You may use numpy and matplotlib functions `np.meshgrid`, `np.linspace`, `reshape`, `contour`.

In [655]:

```
# 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 = f_pred(X2, w_mse)
p = p.reshape(50, -1)

# plot
plt.figure(figsize=(7,5))
plt.scatter(x1[idx_admit], x2[idx_admit], c = 'b', marker = 'o', label = 'Admitted')
plt.scatter(x1[idx_rejec], x2[idx_rejec], c = 'r', marker = 'x', label = 'Rejected')
plt.contour(xx1, xx2, p, levels =[0.5])
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')
plt.legend(loc = 'upper right')
plt.title('Decision boundary using mse')
plt.xlim(25, 103)
plt.ylim(27, 103)
plt.show()
```



In [410]:
```
p = f_pred(X2, w_ce)
p = p.reshape(50, -1)

# plot
plt.figure(figsize=(7,5))
plt.scatter(x1[idx_admit], x2[idx_admit], c = 'b', marker = 'o', label = 'Admitted')
plt.scatter(x1[idx_rejec], x2[idx_rejec], c = 'r', marker = 'x', label = 'Rejected')
plt.contour(xx1, xx2, p, levels =[0.5])
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')
plt.legend(loc = 'upper right')
plt.title('Decision boundary using ce')
plt.xlim(25, 103)
plt.ylim(27, 103)
plt.show()
```

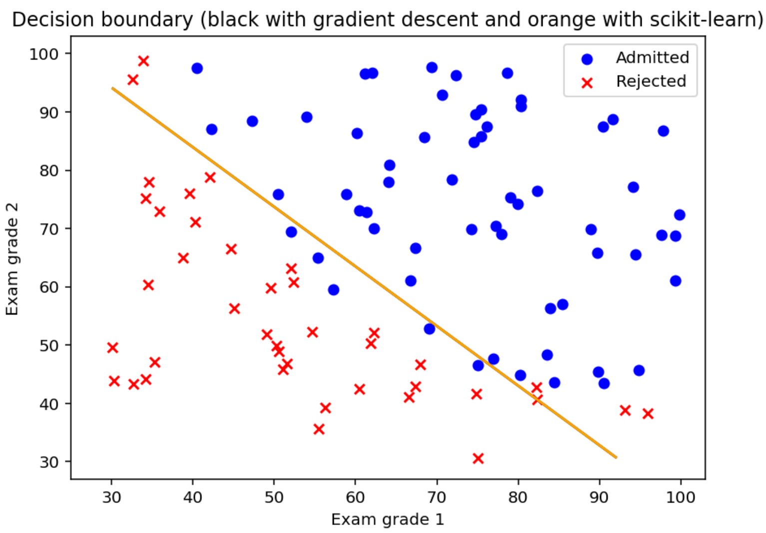


9. Comparison with Scikit-learn logistic regression algorithm with the gradient descent with the cross-entropy loss

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

In [702]:
```
# run logistic regression with scikit-learn
start = time.time()
logreg_sklearn = LogisticRegression(C=1e6)# scikit-learn logistic regression
y = data[:,2][:,None]
logreg_sklearn.fit(X[:, 1:3], y.ravel()) # learn the model parameters

# compute loss value
w_sklearn = np.zeros([3,1])
w_sklearn[0,0] = logreg_sklearn.intercept_
w_sklearn[1:3,0] = logreg_sklearn.coef_

y_pred_sklearn = logreg_sklearn.predict(X[:, 1:3]).reshape(-1, 1)
```

```
loss_sklearn = ce_loss(y_pred_sklearn, y)

# plot
plt.figure(4,figsize=(7,5))
plt.scatter(x1[idx_admit], x2[idx_admit], c = 'b', marker = 'o', label = 'Admitted')
plt.scatter(x1[idx_rejec], x2[idx_rejec], c = 'r', marker = 'x', 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

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

p = f_pred(X2, w_ce)
p = p.reshape(50, -1)
plt.contour(xx1, xx2, p, levels =[0.5], colors = ['black'])

p_sklearn = f_pred(X2, w_sklearn)
p_sklearn = p_sklearn.reshape(50, -1)
plt.contour(xx1, xx2, p_sklearn, levels = [0.5], colors = ['orange']);

plt.title('Decision boundary (black with gradient descent and orange with scikit-learn)')
plt.xlim(25, 103)
plt.ylim(27, 103)
plt.legend()
plt.show()
```

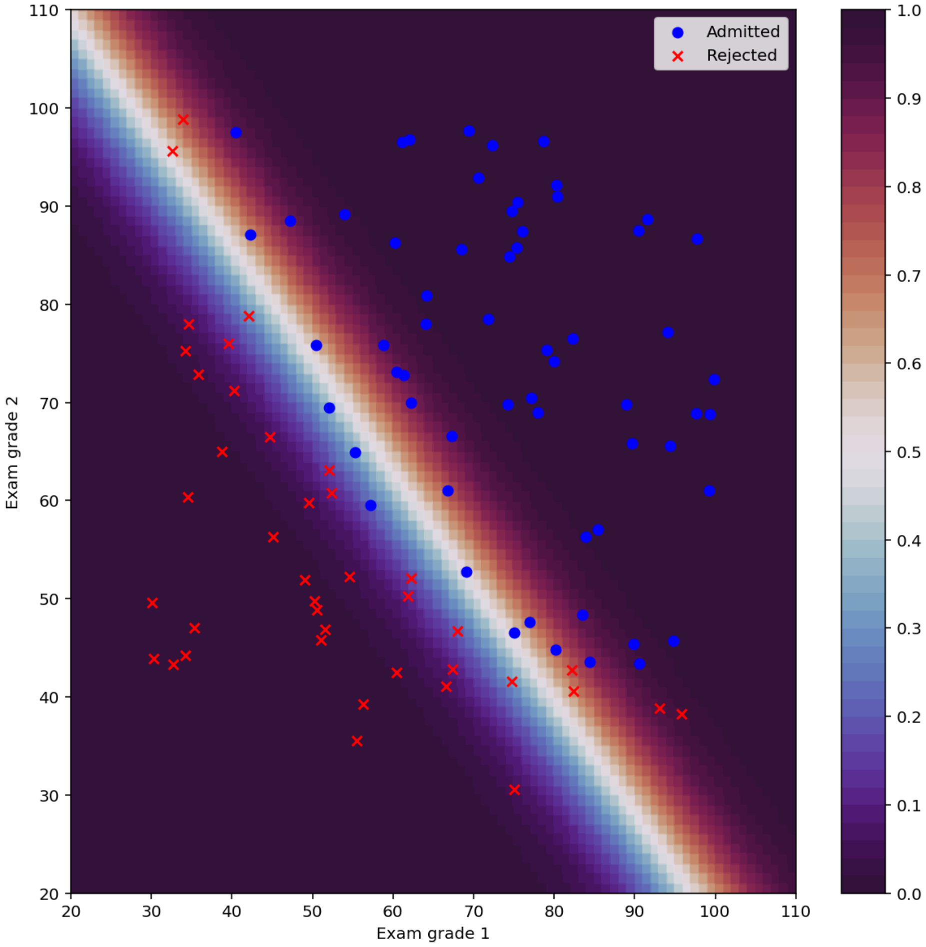


10. Plot the probability map

In [705]:

```
num_a = 500
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.zeros((len(grid_x1), len(grid_x2)))

for i in range(len(grid_x1)):
    for j in range(len(grid_x2)):
            temp_X = np.array([[1, 1, 1]])
            temp_X[0, 1] = grid_x1[i]
            temp_X[0, 2] = grid_x2[j]
            predict_prob = sigmoid(np.dot(temp_X, w_mse))
            Z[j, i] = predict_prob

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

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

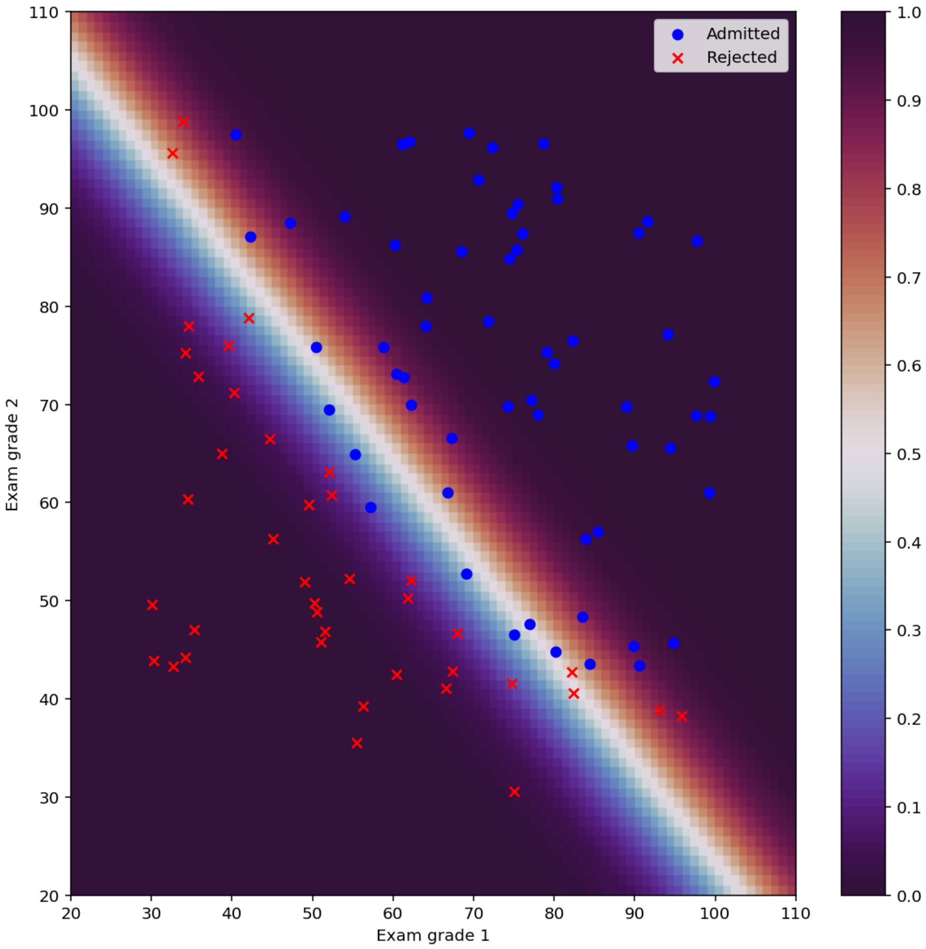
```
ax.set_xlim(20, 110)
ax.set_ylim(20, 110)

cf = ax.contourf(score_x1, score_x2, Z, cmap = 'twilight_shifted', levels = 50)
ax.scatter(x1[idx_admit], x2[idx_admit], c = 'b', marker = 'o', label = 'Admitted')
ax.scatter(x1[idx_rejec], x2[idx_rejec], c = 'r', marker = 'x', label = 'Rejected')
cbar = fig.colorbar(cf, ticks = [0.00, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00],)

cbar.update_ticks()

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



In [707]:

```
num_a = 500
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.zeros((len(grid_x1), len(grid_x2)))

for i in range(len(grid_x1)):
    for j in range(len(grid_x2)):
            temp_X = np.array([[1, 1, 1]])
            temp_X[0, 1] = grid_x1[i]
            temp_X[0, 2] = grid_x2[j]
            predict_prob = sigmoid(np.dot(temp_X, w_ce))
            Z[j, i] = predict_prob

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

ax = fig.add_subplot(111)
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, Z, cmap = 'twilight_shifted', levels = 100)
ax.scatter(x1[idx_admit], x2[idx_admit], c = 'b', marker = 'o', label = 'Admitted')
```

```
ax.scatter(x1[idx_rejec], x2[idx_rejec], c = 'r', marker = 'x', label = 'Rejected')
cbar = fig.colorbar(cf, ticks = [0.00, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00])

cbar.update_ticks()

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

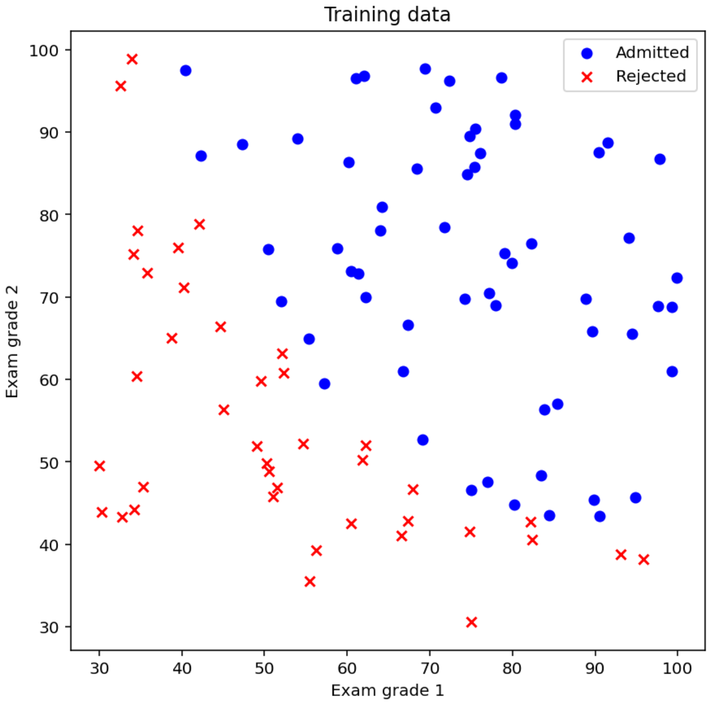


Output results

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

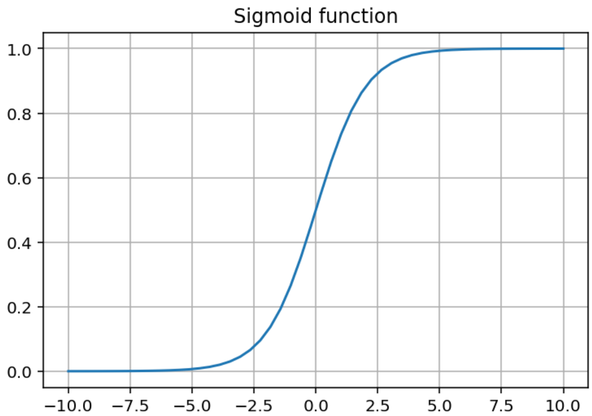
In [17]:

```
plt.figure(figsize = (7, 7))
plt.scatter(x1[idx_admit], x2[idx_admit], c = 'b', marker = 'o', label = 'Admitted')
plt.scatter(x1[idx_rejec], x2[idx_rejec], c = 'r', marker = 'x', label = 'Rejected' )
plt.title('Training data')
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')
plt.legend(loc = 'upper right')
plt.show()
```



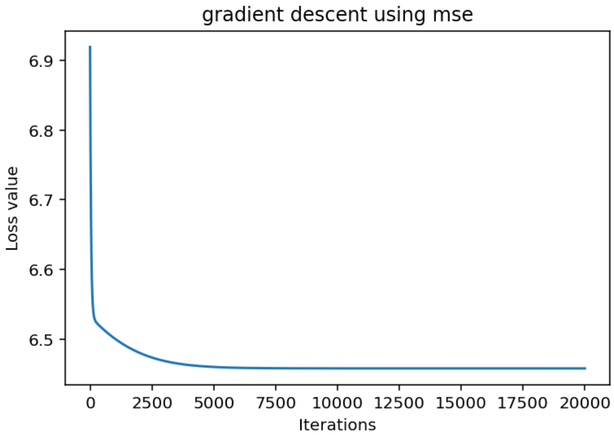
2. Plot the sigmoid function (1pt)

In [19]:
```
# plot
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)

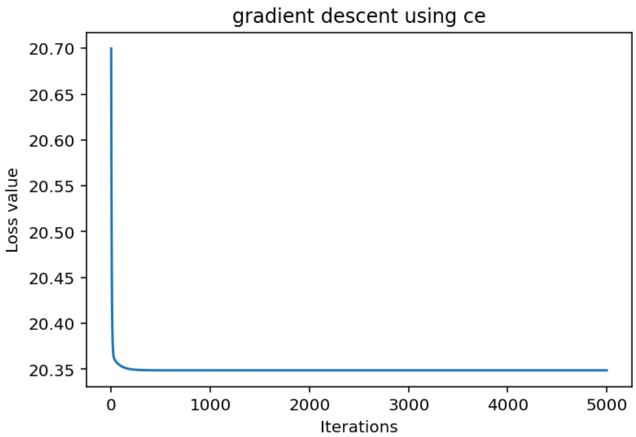
In [695]:
```
start = time.time()
w_init = np.array([-25, 0.2, 0.2])[:,None]
tau = 0.0001; max_iter = 20000
w_mse, L_iters, w_iters = grad_desc_mse(X,y,w_init,tau,max_iter)

# plot
plt.figure(3)
plt.plot(L_iters)
plt.title('gradient descent using mse')
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 [694]:

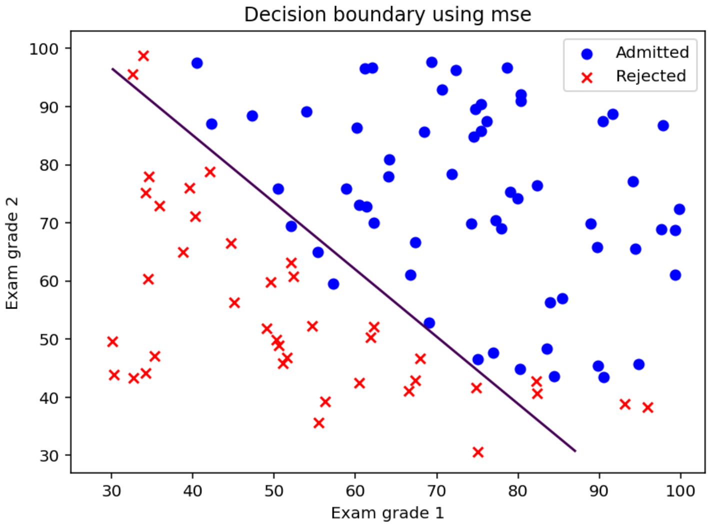
```
start = time.time()
w_init = np.array([-25, 0.2, 0.2])[:,None]

tau = 0.0001; max_iter = 5000
w_ce, L_iters, w_iters = grad_desc_ce(X,y,w_init,tau,max_iter)

# plot
plt.figure(3)
plt.plot(L_iters)
plt.title('gradient descent using ce')
plt.xlabel('Iterations')
plt.ylabel('Loss value')
plt.show()
```



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

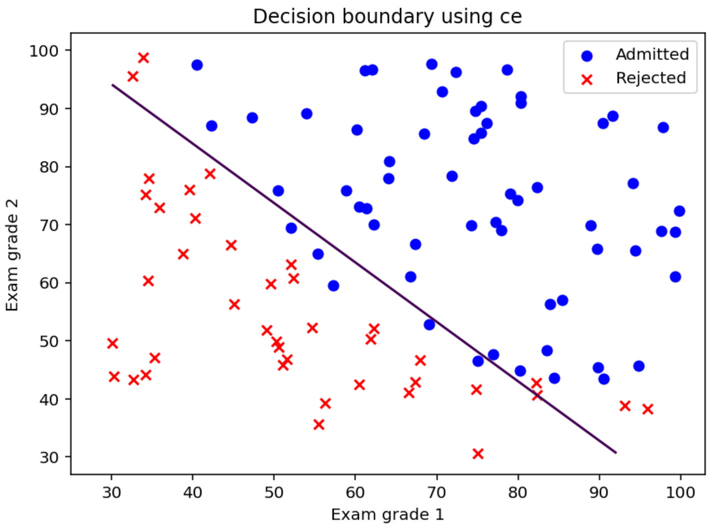
In [696]:

```
p = f_pred(X2, w_mse)
p = p.reshape(50, -1)

# plot
plt.figure(figsize=(7,5))
plt.scatter(x1[idx_admit], x2[idx_admit], c = 'b', marker = 'o', label = 'Admitted')
plt.scatter(x1[idx_rejec], x2[idx_rejec], c = 'r', marker = 'x', label = 'Rejected')
plt.contour(xx1, xx2, p, levels =[0.5])
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')
plt.legend(loc = 'upper right')
plt.title('Decision boundary using mse')
plt.xlim(25, 103)
plt.ylim(27, 103)
plt.show()
```



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

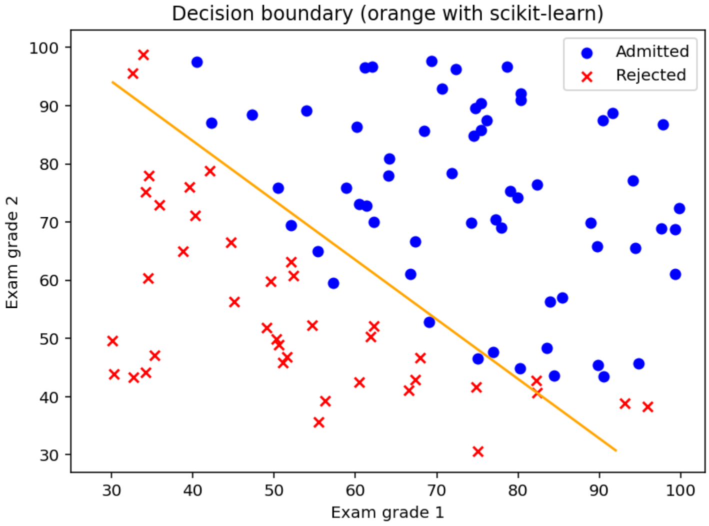
In [697]:
```
p = f_pred(X2, w_ce)
p = p.reshape(50, -1)

# plot
plt.figure(figsize=(7,5))
plt.scatter(x1[idx_admit], x2[idx_admit], c = 'b', marker = 'o', label = 'Admitted')
plt.scatter(x1[idx_rejec], x2[idx_rejec], c = 'r', marker = 'x', label = 'Rejected')
plt.contour(xx1, xx2, p, levels =[0.5])
plt.xlabel('Exam grade 1')
plt.ylabel('Exam grade 2')
plt.legend(loc = 'upper right')
plt.title('Decision boundary using ce')
plt.xlim(25, 103)
plt.ylim(27, 103)
plt.show()
```



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

In [698]:
```
plt.figure(4,figsize=(7,5))
plt.scatter(x1[idx_admit], x2[idx_admit], c = 'b', marker = 'o', label = 'Admitted')
plt.scatter(x1[idx_rejec], x2[idx_rejec], c = 'r', marker = 'x', 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

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

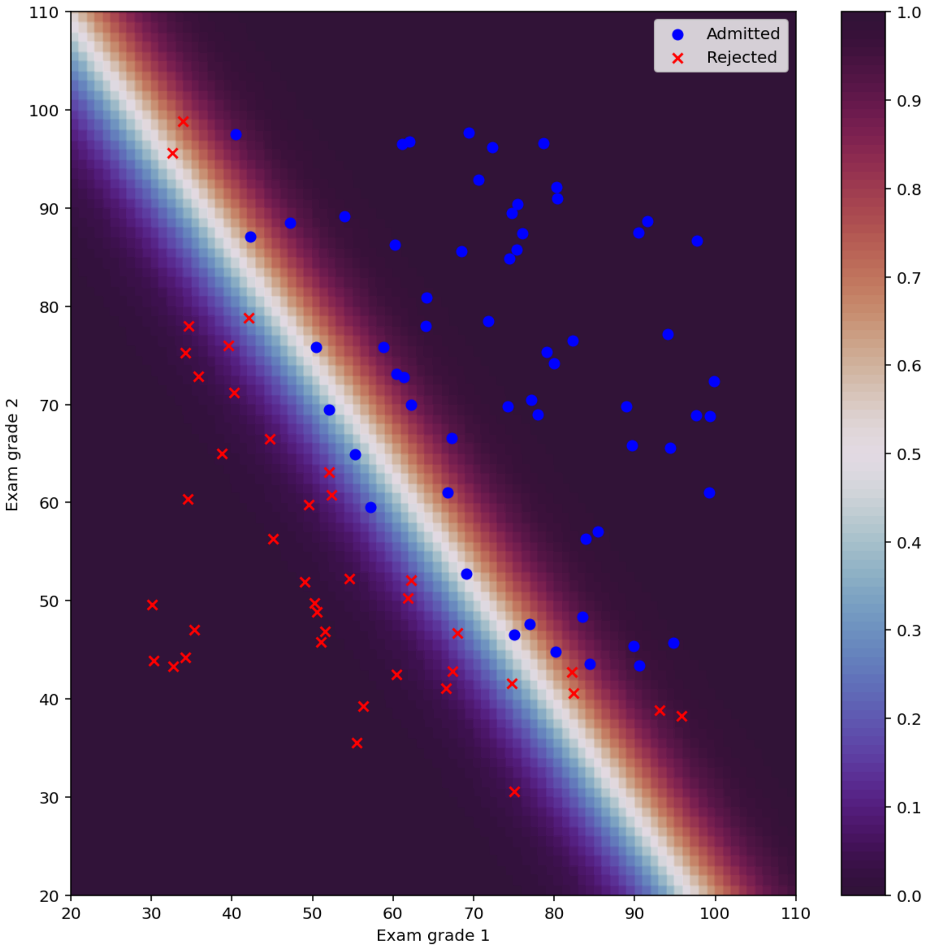
```python
X2[:,2] = xx2.reshape(-1)

p_sklearn = f_pred(X2, w_sklearn)
p_sklearn = p_sklearn.reshape(50, -1)
plt.contour(xx1, xx2, p_sklearn, levels = [0.5], colors = ['orange']);

plt.title('Decision boundary (orange with scikit-learn)')
plt.xlim(25, 103)
plt.ylim(27, 103)
plt.legend()
plt.show()
```



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

In [706]:

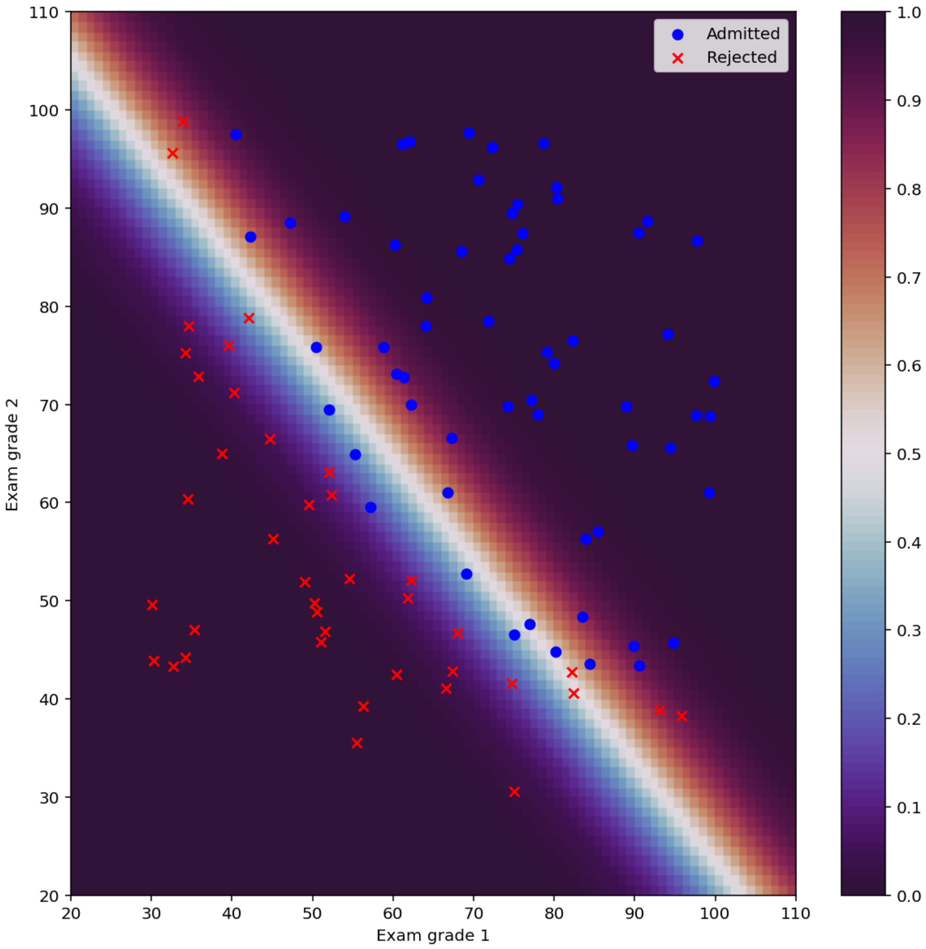
```python
# actual plotting example
fig = plt.figure(figsize=(10,10))

ax = fig.add_subplot(111)
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, Z, cmap = 'twilight_shifted', levels = 100)
ax.scatter(x1[idx_admit], x2[idx_admit], c = 'b', marker = 'o', label = 'Admitted')
ax.scatter(x1[idx_rejec], x2[idx_rejec], c = 'r', marker = 'x', label = 'Rejected')
cbar = fig.colorbar(cf, ticks = [0.00, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00],)

cbar.update_ticks()

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



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

In [708]:

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

ax = fig.add_subplot(111)
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, Z, cmap = 'twilight_shifted', levels = 100)
ax.scatter(x1[idx_admit], x2[idx_admit], c = 'b', marker = 'o', label = 'Admitted')
ax.scatter(x1[idx_rejec], x2[idx_rejec], c = 'r', marker = 'x', label = 'Rejected')
cbar = fig.colorbar(cf, ticks = [0.00, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00])

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
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

In [ ]: