```
import numpy as np
import matplotlib.pyplot as plt

##-------------------------------------Part(a)
x = np.load("fashion_mnist_images.npy")
y = np.load("fashion_mnist_labels.npy")
d, n= x.shape
i = 1#Index of the image to be visualized
plt.imshow(np.reshape(x[:,i], (int(np.sqrt(d)),int(np.sqrt(d)))), cmap="
    Greys")
plt.show()

# reshape y to shape = (n,1)
y = y.reshape(-1,1)

# reshape x to shape = (n,d)
x_1 = np.transpose(x)

# append 1 to get x_tilde
x_2 = np.concatenate((np.ones(n).reshape(-1,1), x_1), axis = 1)

# split the training data and test data
X_train = x_2[:5000]
X_test = x_2[5000:]
y_train = y[:5000]
y_test = y[5000:]

# initialize Theta
theta = np.zeros(785)


# Define the loss function
def loss_function(X, y, lamb, theta):
    loss = 0
    w = theta.copy()

    for i in range(X.shape[0]):
        # Compute the loss for each data point
        loss += np.log(1 + np.exp(-y[i] * np.dot(theta, X[i])))

    w[0] = 0  # Exclude the bias term from regularization

    # Add the regularization term to the loss
    loss += lamb * np.dot(w, w)

    return loss.item()

# Define the function of computing Gradient
def gradient(X, y, lamb, theta):
```

```
    grad = 0
    w = theta.copy()

    for i in range(X.shape[0]):
        grad += -y[i] * np.exp(-y[i] * np.dot(theta, X[i]))/(1 + np.exp(-y[
            i] * np.dot(theta, X[i]))) * X[i]

    w[0] = 0

    grad = grad + 2 * lamb * w

    return grad

# Define the function of computing Hessian
def hessian(X, y, lamb, theta):
    hess = np.zeros((785, 785))
    w = theta.copy()
# Compute the Hessian according to the formula
    for i in range(X.shape[0]):
        hess += np.exp(y[i] * np.dot(theta, X[i]))/((1 + np.exp(y[i] * np.
            dot(theta, X[i])))**2) * np.dot(X[i].reshape(785, 1), X[i].
            reshape(785, 1).T)

    # Define the matrix for w
    I_w = np.identity(785)
    I_w[0, 0] = 0

    hess = hess + 2 * lamb * I_w
    return hess


## Algorithm of Newton-Raphson
def optimize(X, y, lamb, theta, epsilon, max_iter):
    iter = 0
    stop = False
    loss = loss_function(X, y, lamb, theta)

    while not stop and iter < max_iter:
        grad_now = gradient(X, y, lamb, theta)
        hess_now = hessian(X, y, lamb, theta)

        theta_1 = theta - np.dot(np.linalg.inv(hess_now), grad_now)
        loss_new = loss_function(X, y, lamb, theta_1)
# If the relative error is greater than the epsilon, then stop the
    algorithm and return num of iteration, parameter, and loss
        if np.abs(loss_new - loss)/loss > epsilon:
            loss = loss_new
            iter = iter + 1
            theta = theta_1
```

```
        else:
            stop = True

    return iter, theta, loss

result = optimize(X_train, y_train, 1, theta, 1e-6, 20)

result

## Extract the parameter theta
theta_final = result[1]

# Plug the theta into the logistic function
def prob_test(X, y):
    final_result = []
    for i in range(X.shape[0]):
        #Expression of logistic regression
        probs = 1/(1 + np.exp(-np.dot(theta_final, X[i])))
        final_result.append(probs)

    return final_result

def trans_prob(input_list):
    indicator = []
    for i in range(1000):
        if input_list[i] >= 0.5:
            indicator.append(1)
        else:
            indicator.append(-1)
    return indicator

# Get the final accuracy
def valid(X, Y):
    accuracy = []
    for i in range(1000):
        if X[i] == Y[i]:
            accuracy.append(1)
        else:
            accuracy.append(0)
    return sum(accuracy)/1000


# Get the probability of label 1 for test set
final_prob = prob_test(X_test, y_test)
final_label = trans_prob(final_prob)

y_test_result = y_test.ravel().tolist()
test_error = 1 - valid(final_label, y_test_result)
print("The␣test␣error␣is:", test_error)
```

```
###--------------------------------------Part(b)
# I first substract 0.5 for each element in the list, and then transform it
    into an array
abs_distance_half = np.array([abs(x - 0.5) for x in final_prob])

# Get the rank for each element
sorted_index = np.argsort(abs_distance_half)


# Then we will get the misclassified image for the top 20 images and get
   its image
misclassified_index = []
for i in range (sorted_index.shape[0]):
    if y_test_result[sorted_index[i]] != final_label[sorted_index[i]] and
       len(misclassified_index) < 20:
        misclassified_index.append(sorted_index[i])
    else:
        pass


for i in range(20):
    plt.subplot(4, 5, i+1)
    #Here we need to ignore the first element for each data because the
       first element is always 1
    plt.imshow(X_test[misclassified_index[i]][1:].reshape((int(np.sqrt(784)
       ),int(np.sqrt(784)))), cmap="Greys")
    plt.title(y_test[misclassified_index[i]][0])
    plt.axis('off')
# adjust the distance between different plots
plt.subplots_adjust(left=0.1, bottom=0.1, right=0.9, top=0.9, wspace=0.4,
   hspace=0.6)

plt.show()
```