Homework 3: Regression Theory

CSE6242 - Data and Visual Analytics - Summer 2018

Due: Sunday, July 8, 2018 at 11:59 PM UTC-12:00 on T-Square

hyang390, 903320189

1. Data Preprocessing [50 points]

```
mnist_train = read.csv('mnist/mnist_train.csv', header=FALSE)
mnist_test = read.csv('mnist/mnist_test.csv', header=FALSE)

train_0_1 = mnist_train[,((mnist_train[785,] == 0) | (mnist_train[785,] == 1))]
train_3_5 = mnist_train[,((mnist_train[785,] == 3) | (mnist_train[785,] == 5))]

test_0_1 = mnist_test[,((mnist_test[785,] == 0) | (mnist_test[785,] == 1))]
test_3_5 = mnist_test[,((mnist_test[785,] == 3) | (mnist_test[785,] == 5))]


print(sprintf("Dimension of train_0_1 are %i by %i.", dim(train_0_1)[1], dim(train_0_1)[2]))
```

```
## [1] "Dimension of train_0_1 are 785 by 12665."
```

```
print(sprintf("Dimension of train_3_5 are %i by %i.", dim(train_3_5)[1], dim(train_3_5)[2]))
```

```
## [1] "Dimension of train_3_5 are 785 by 11552."
```

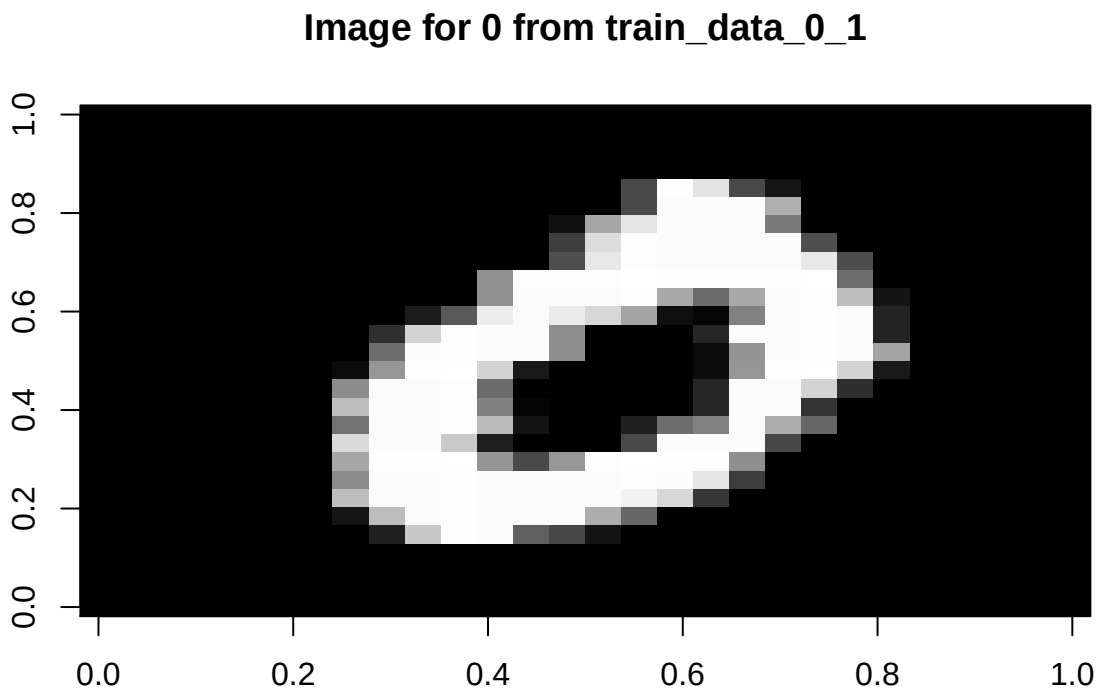
```
print(sprintf("Dimension of test_0_1 are %i by %i.", dim(test_0_1)[1], dim(test_0_1)[2]))
```

```
## [1] "Dimension of test_0_1 are 785 by 2115."
```

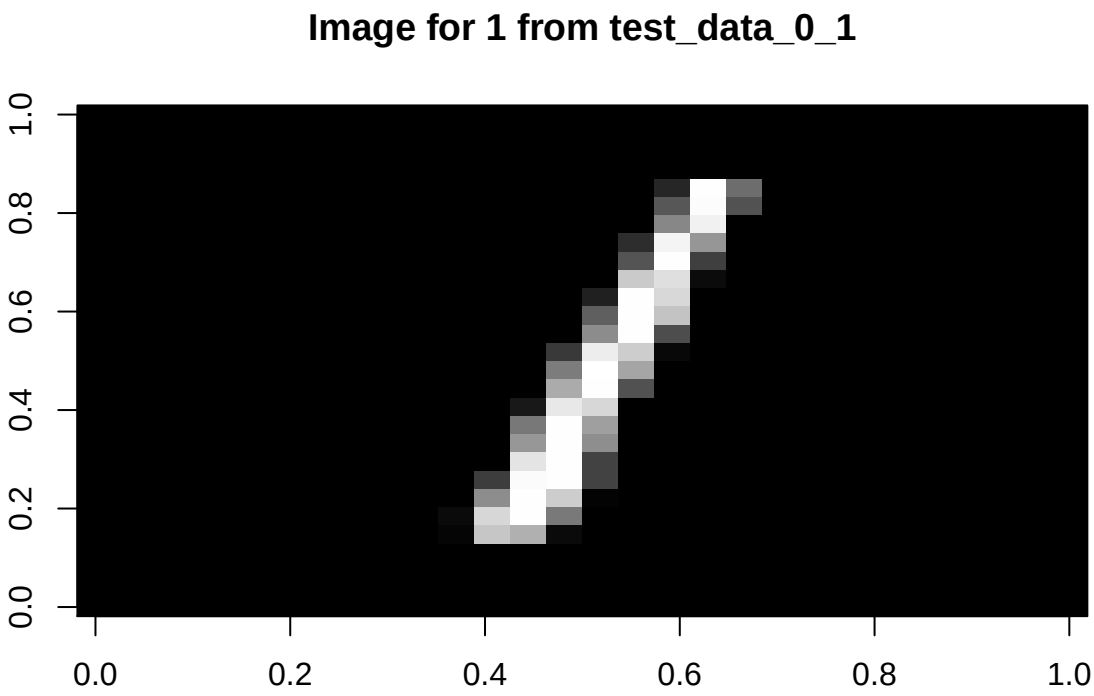
```
print(sprintf("Dimension of test_3_5 are %i by %i.", dim(test_3_5)[1], dim(test_3_5)[2]))
```

```
## [1] "Dimension of test_3_5 are 785 by 1902."
```

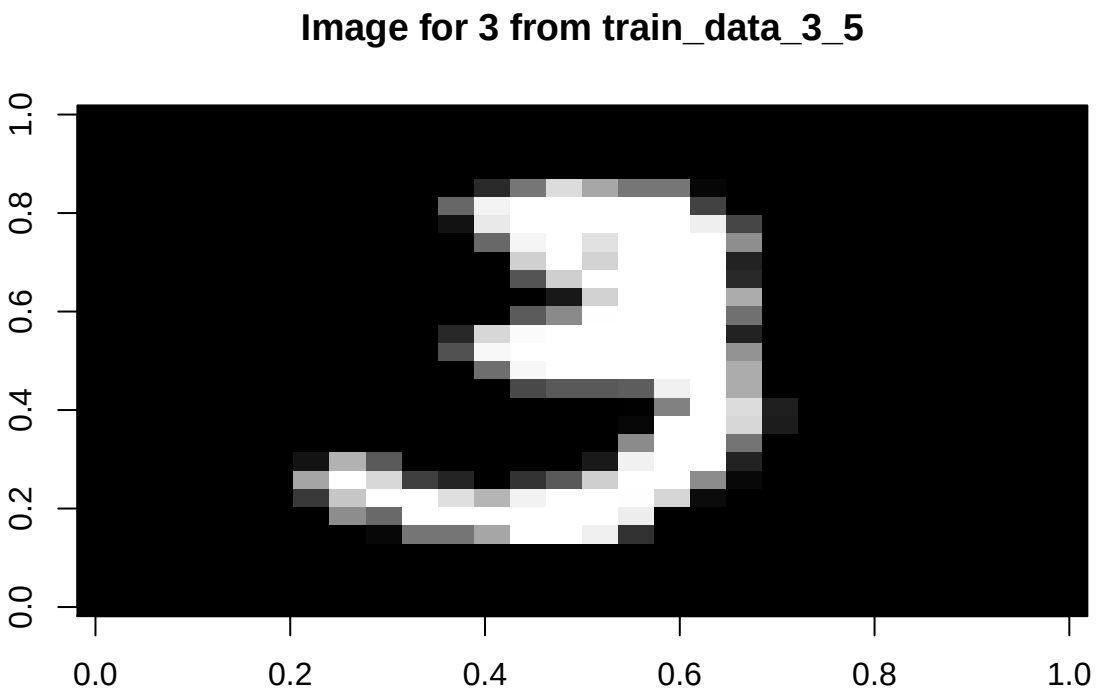
```
train_data_0_1 = train_0_1[-785, ]
train_data_3_5 = train_3_5[-785, ]
train_labels_0_1 = train_0_1[785, ]
train_labels_3_5 = train_3_5[785, ]

test_data_0_1 = test_0_1[-785, ]
test_data_3_5 = test_3_5[-785, ]
test_labels_0_1 = test_0_1[785, ]
test_labels_3_5 = test_3_5[785, ]

rotate = function(x) t(apply(x, 2, rev))

image(rotate(matrix(data=train_data_0_1[ , 4], nrow=28, ncol=28)), col=gray(0:255/255),
      main="Image for 0 from train_data_0_1")
```



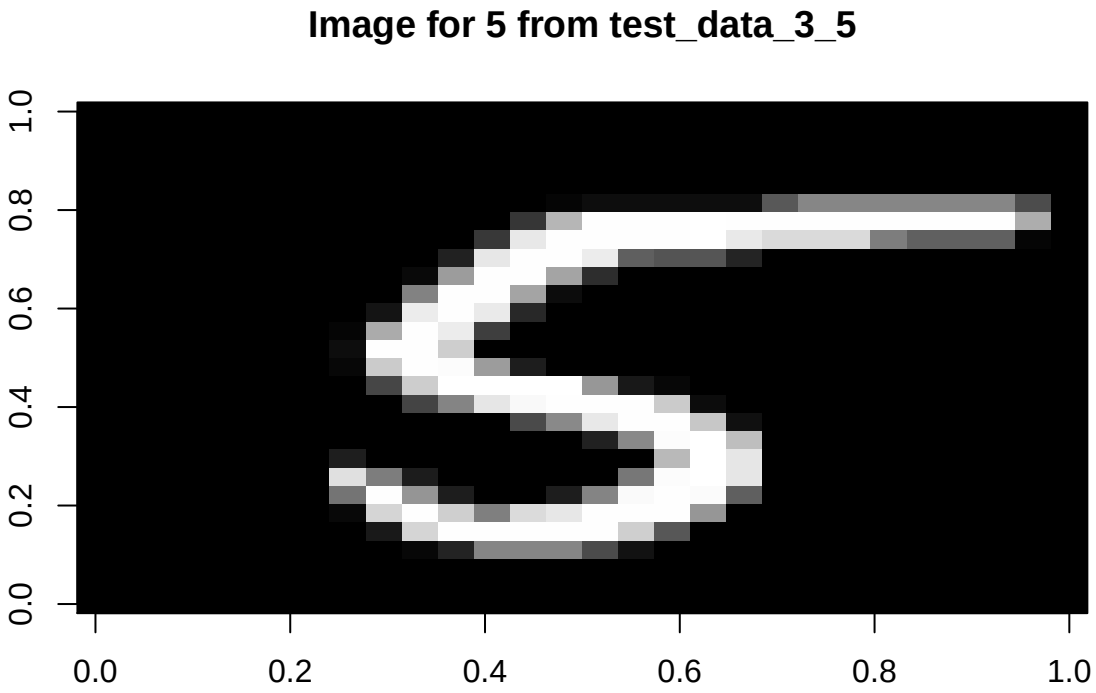

```
image(rotate(matrix(data=test_data_0_1[ , 981], nrow=28, ncol=28)), col=gray(0:255/255),
      main="Image for 1 from test_data_0_1")
```




```
image(rotate(matrix(data=train_data_3_5[ , 2], nrow=28, ncol=28)), col=gray(0:255/255),
      main="Image for 3 from train_data_3_5")
```




```
image(rotate(matrix(data=test_data_3_5[ , 1050], nrow=28, ncol=28)), col=gray(0:255/255),
      main="Image for 5 from test_data_3_5")
```



2. Theory [50 points]

We assume that

$$y_i \in \{-1, 1\}$$

Then, from the lectures and notes, we know the loss function for logistic regression in that range is (added the negative sign in front according to errata and Piazza instructions):

$$L(\theta) = \sum_{i=1}^{n} \log\left(1 + \exp(-y^{(i)}\langle\theta, x^{(i)}\rangle)\right)$$

To derive the gradient of the loss function with respect to model parameters, we take the partial derivative

$$\frac{\partial L(\theta)}{\partial \theta_j}$$

By the chain rule and the rule of summation of derivatives, we know that

$$\frac{\partial L(\theta)}{\partial \theta_j} = \sum_{i=1}^{n} \frac{1}{1 + \exp(-y^{(i)}\langle\theta, x^{(i)}\rangle)} \frac{\partial}{\partial \theta_j}(1 + \exp(-y^{(i)}\langle\theta, x^{(i)}\rangle)$$

$$= \sum_{i=1}^{n} \frac{1}{1 + \exp(-y^{(i)}\langle\theta, x^{(i)}\rangle)} \exp(-y^{(i)}\langle\theta, x^{(i)}\rangle) \frac{\partial}{\partial \theta_j}(-y^{(i)}\langle\theta, x^{(i)}\rangle)$$

We know that the derivative of of the term

$$(-y^{(i)}\langle\theta, x^{(i)}\rangle)$$

is as follows:

$$\frac{\partial}{\partial \theta_j}(-y^{(i)}\langle \theta, x^{(i)}\rangle) = \frac{\partial(-y^{(i)}\theta_1 x_1^{(i)} + ... + -y^{(i)}\theta_d x_d^{(i)})}{\partial \theta_j} = -y^{(i)}x_j^{(i)}$$

Putting this back into the above, we have

$$\frac{\partial L(\theta)}{\partial \theta_j} = \sum_{i=1}^{n} \frac{-y^{(i)}x_j^{(i)}}{1+\exp(-y^{(i)}\langle \theta, x^{(i)}\rangle)} \exp(-y^{(i)}\langle \theta, x^{(i)}\rangle)$$

By dividing

$$\exp(-y^{(i)}\langle \theta, x^{(i)}\rangle)$$

top and bottom, we get

$$\frac{\partial L(\theta)}{\partial \theta_j} = \sum_{i=1}^{n} \frac{-y^{(i)}x_j^{(i)}}{1+\exp(y^{(i)}\langle \theta, x^{(i)}\rangle)}$$

Plugging the above into the update rule for stochastic gradient descent, we have the following:

$$\theta_j \leftarrow \theta_j + \alpha \sum_{i=1}^{n} \frac{y^{(i)}x_j^{(i)}}{1+\exp(y^{(i)}\langle \theta, x^{(i)}\rangle)}$$

We can express the following in pseudocode as follows:

```
threshold <= 0.001 #threshold for convergence, could be different value
alpha <= 0.8 #alpha, learning rate
#assume samples for x is n by d, and y is of size n

#initialize thetas to random values
theta = [] #vector of size d
for (j in 0:d) {
  theta[j] = random(0, 1) #initialize thetas to random values between 0 and 1
}

##repeat until theta converges (for all from 0 to d)
while derivative_theta > threshold {
  z = []
  #calculate the inner products of theta and x
  for (i in 1:n) {
    for (j in 0:d) {
      z[i] = sum(theta[j]*x[i][j]) #for j from 0 to d
    }
  }
  #summation of the resultant partial derivative
  for (i in 1:n){
    delta = 0
    for (j in 1:d) {
      delta = delta + ((y[i] * x[i][j]) / (1 + exp(y[i] * z[i])))
    }
  }
  theta = theta + alpha * delta #assuming vectorized code, for 0 to d for theta_j
  derivative_theta = alpha * delta
```

```
}

return theta
```

The runtime of the above pseducode is about

$$O(nd)$$

per epoch, as each iteration through n requires an iteration through the dimensions d as well.