Homework 4: Regression Practice

CSE6242 - Data and Visual Analytics - Summer 2018

Due: Sunday, July 22, 2018 at 11:59 PM UTC-12:00 on T-Square

hyang390, 903320189

Data Preprocessing

```
#setwd("C:/Users/Batmachine/Dropbox/OMSCS/CSE6242 - DVA/Assignments/hw4")
library("ggplot2")
library("reshape2")
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))]


train_data_0_1 = as.matrix(t(train_0_1[-785, ]))
train_data_3_5 = as.matrix(t(train_3_5[-785, ]))
train_labels_0_1 = as.matrix(t(train_0_1[785, ]))
train_labels_3_5 = as.matrix(t(train_3_5[785, ]))

test_data_0_1 = as.matrix(t(test_0_1[-785, ]))
test_data_3_5 = as.matrix(t(test_3_5[-785, ]))
test_labels_0_1 = as.matrix(t(test_0_1[785, ]))
test_labels_3_5 = as.matrix(t(test_3_5[785, ]))

train_data_0_1 = cbind(1, train_data_0_1)
train_data_3_5 = cbind(1, train_data_3_5)
test_data_0_1 = cbind(1, test_data_0_1)
test_data_3_5 = cbind(1, test_data_3_5)

train_labels_0_1 = ifelse(train_labels_0_1 == 0, -1, 1)
train_labels_3_5 = ifelse(train_labels_3_5 == 3, -1, 1)
test_labels_0_1 = ifelse(test_labels_0_1 == 0, -1, 1)
test_labels_3_5 = ifelse(test_labels_3_5 == 3, -1, 1)
```

1. Implementation [35 points]

In my implementation, I followed the algorithms I laid out in my previous homework 3 in combination with the exemplary homeworks. I first mapped all labels to -1 and 1, with the smaller number being -1 and the other being 1. Then, initialized theta to random values of a uniform distribution between 0 and 1. For shuffling in each epoch, I first combined the data and labels using "cbind", then shuffled them and unbound them to retrieve back the data and labels. This ensures that they are all shuffled in the same order. All other steps of the algorithm followed the derivations shown in the previous homework. As for convergence

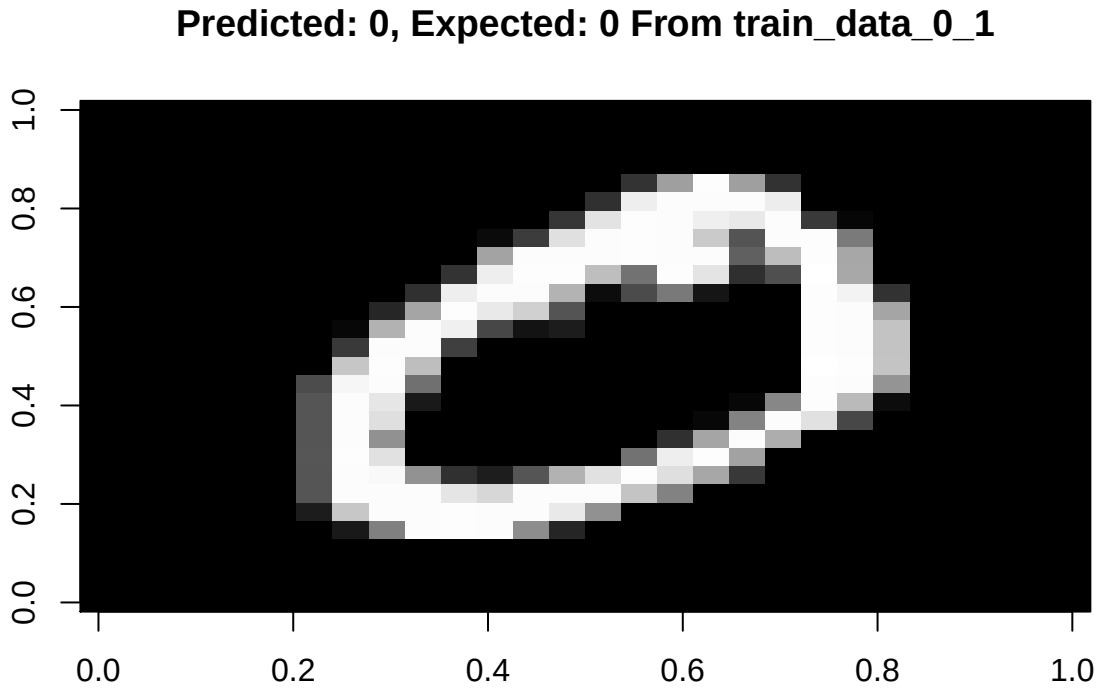
criteria, I used the difference of the loss function to check if the algorithm has converged. If after 20 epochs without convergence, the algorithm will just return the thetas to shorten runtimes. Please note that due to the fact that my accuracy for training data 0/1 is 100%, I was not able to include the incorrect images for the training set 0/1.

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

predict = function(theta, data) {
  probability = 1/(1+exp(-(as.matrix(data) %*% theta)))
  labels = ifelse(probability > 0.5, 1, -1)
  return (labels)
}

bind_and_shuffle = function(data, labels) {
  bound_data_labels = cbind(data, labels)
  bound_data_labels_shuffled = bound_data_labels[sample(nrow(bound_data_labels)), ]
  return (bound_data_labels_shuffled)
}

accuracy = function(predicted_labels, true_labels) {
  acc = sum(predicted_labels == true_labels) / dim(true_labels)[1]
  return (acc)
}

train = function(data, labels, alpha){
  threshold = 0.01
  epochs = 20
  theta = as.matrix(runif(dim(data)[2], 0, 1))
  loss_old = Inf
  product = data %*% theta
  loss_new = sum(1 + exp(-(labels * product)))

  for(epoch in 1:epochs) {
    shuffled_data_labels = bind_and_shuffle(data, labels)
    data = shuffled_data_labels[, -786]
    labels = as.matrix(shuffled_data_labels[, 786])

    for (i in 1:dim(data)[1])
    {
      x_i = t(as.matrix(data[i,]))
      y_i = t(as.matrix(labels[i,]))
      product = x_i %*% theta
      delta = t(x_i) %*% y_i / as.numeric(1 + exp(y_i * product))
      theta = theta + alpha * delta
    }

    loss_old = loss_new
    product = data %*% theta
    loss_new = sum(1 + exp(-(labels * product)))
    if (abs(loss_new - loss_old) <= threshold) {
      return (theta)
    }
  }
  return (theta)
```

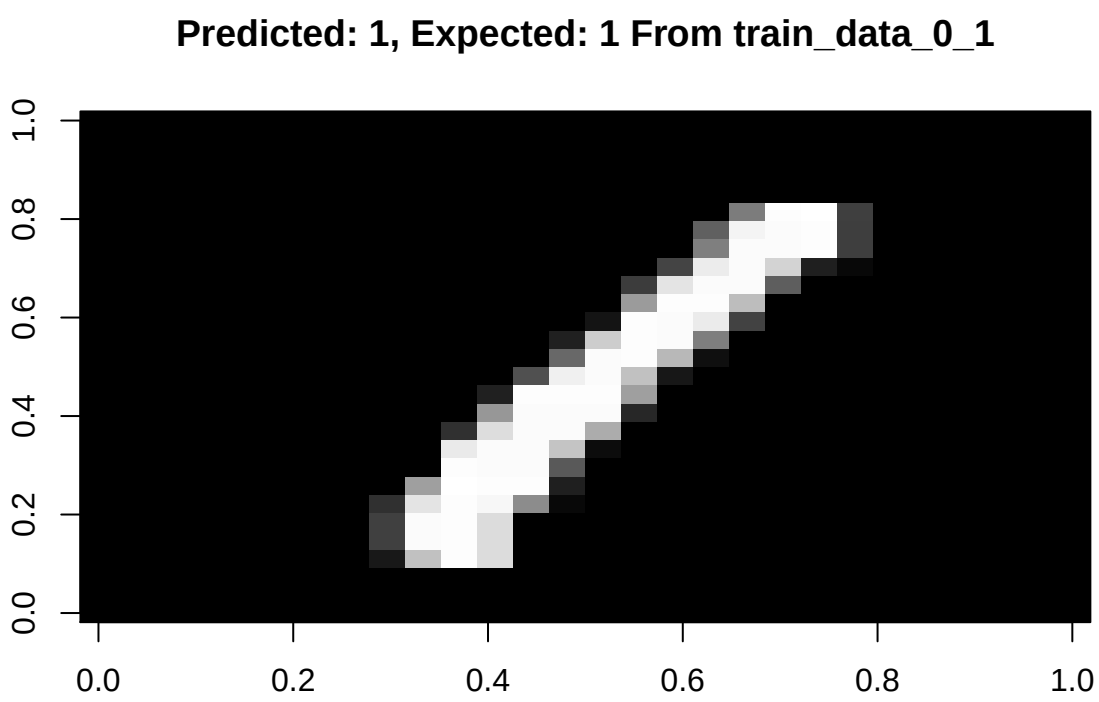
```
}


##### Run train() on training sets #####
theta_01 = train(train_data_0_1, train_labels_0_1, 0.2)
prediction_01 = predict(theta_01, train_data_0_1)
acc_01 = accuracy(prediction_01, train_labels_0_1)
correct_01 = which(train_labels_0_1 == prediction_01)
incorrect_01 = which(train_labels_0_1 != prediction_01)

theta_35 = train(train_data_3_5, train_labels_3_5, 0.6)
prediction_35 = predict(theta_35, train_data_3_5)
acc_35 = accuracy(prediction_35, train_labels_3_5)
correct_35 = which(train_labels_3_5 == prediction_35)
incorrect_35 = which(train_labels_3_5 != prediction_35)


print(sprintf("Accuracy of the training set 0/1 and 3/5 is %f and %f respectively. Therefore no incorre
```

```
## [1] "Accuracy of the training set 0/1 and 3/5 is 1.000000 and 0.958276 respectively. Therefore no in
```

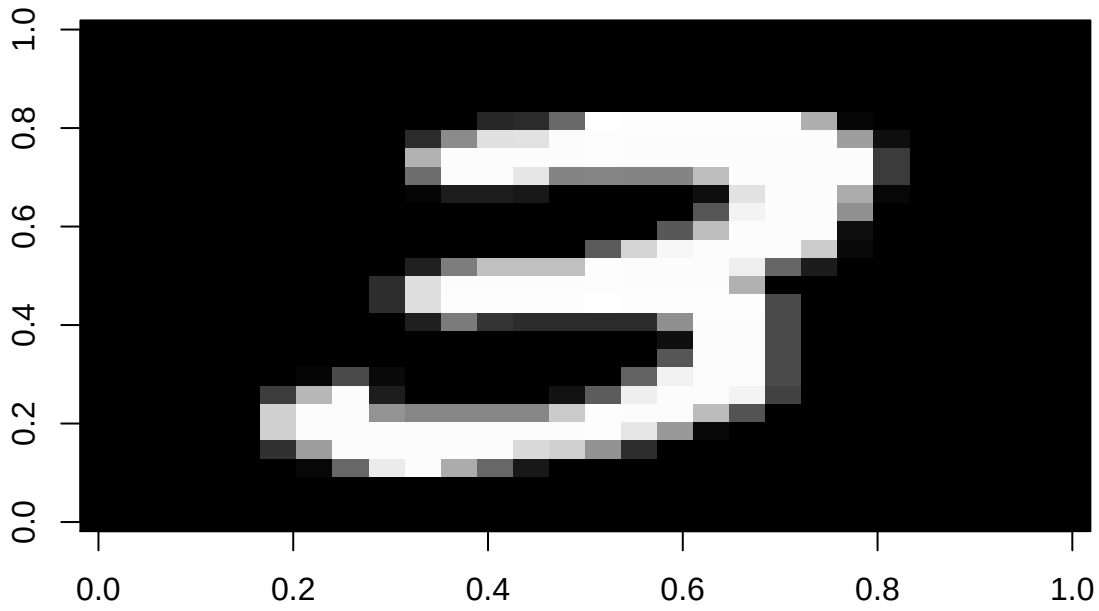
```
image(rotate(matrix(data=t(train_data_0_1)[2:785, correct_01[1]], nrow=28, ncol=28)), col=gray(0:255/255
      main="Predicted: 0, Expected: 0 From train_data_0_1")
```



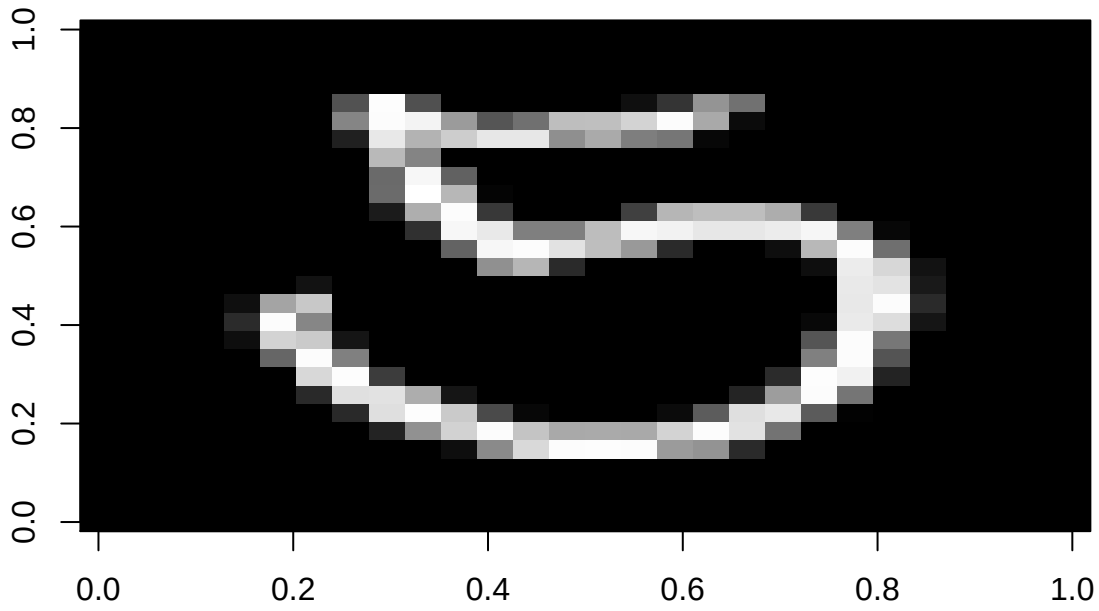

```
image(rotate(matrix(data=t(train_data_0_1)[2:785, correct_01[5924]], nrow=28, ncol=28)), col=gray(0:255/
      main="Predicted: 1, Expected: 1 From train_data_0_1")
```



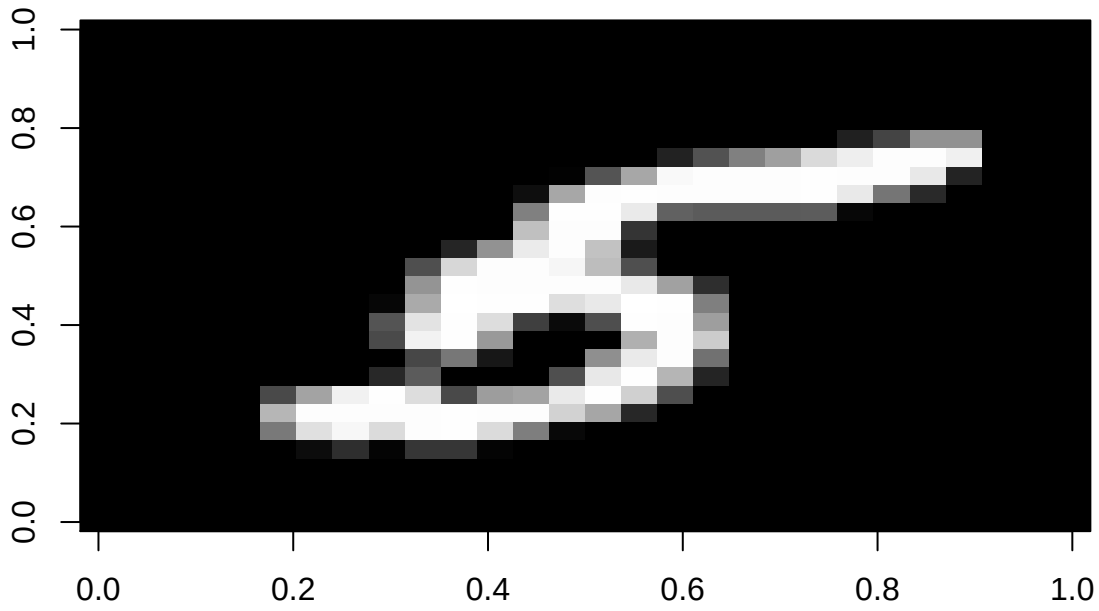

```
image(rotate(matrix(data=t(train_data_3_5)[2:785, correct_35[1]], nrow=28, ncol=28)), col=gray(0:255/255
      main="Predicted: 3, Expected: 3 From train_data_3_5")
```



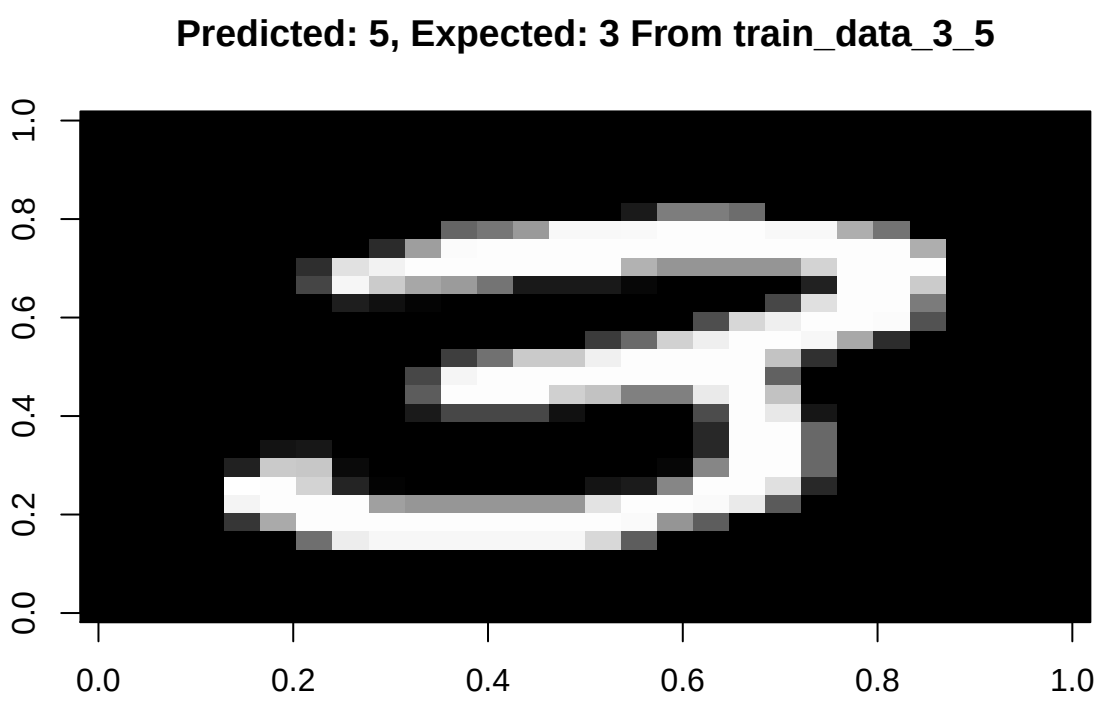

```
image(rotate(matrix(data=t(train_data_3_5)[2:785, incorrect_35[386]], nrow=28, ncol=28)), col=gray(0:255
      main="Predicted: 3, Expected: 5 From train_data_3_5")
```




```
image(rotate(matrix(data=t(train_data_3_5)[2:785, correct_35[6233]], nrow=28, ncol=28)), col=gray(0:255/
      main="Predicted: 5, Expected: 5 From train_data_3_5")
```




```
image(rotate(matrix(data=t(train_data_3_5)[2:785, incorrect_35[1]], nrow=28, ncol=28)), col=gray(0:255/
      main="Predicted: 5, Expected: 3 From train_data_3_5")
```



2. Modelling [35 points]

```
accuracy = function(predicted_labels, true_labels) {
  acc = sum(predicted_labels == true_labels) / dim(true_labels)[1]
  return (acc)
}

model = function(train_data, train_labels, test_data, test_labels, alpha) {
  theta = train(train_data, train_labels, alpha)
  prediction_train = predict(theta, train_data)
  acc_train = accuracy(prediction_train, train_labels)

  prediction_test = predict(theta, test_data)
  acc_test = accuracy(prediction_test, test_labels)
  return (list(theta, acc_train, acc_test))
}

learning_rates = seq(0.1, 1.0, by = 0.1)
results_01 = list()
results_35 = list()

for (i in 1:length(learning_rates)) {
  exp_01 = model(train_data_0_1, train_labels_0_1, test_data_0_1, test_labels_0_1, learning_rates[i])
  results_01[[i]] = c('dataset_0_1', learning_rates[i], exp_01[[2]], exp_01[[3]])
```

```
  exp_35 = model(train_data_3_5, train_labels_3_5, test_data_3_5, test_labels_3_5, learning_rates[i])
  results_35[[i]] = c('dataset_3_5', learning_rates[i], exp_35[[2]], exp_35[[3]])
}

total_results = append(results_01, results_35)
results_01 = as.data.frame(do.call(rbind, results_01))
results_35 = as.data.frame(do.call(rbind, results_35))
results_df = as.data.frame(do.call(rbind, total_results))
colnames(results_df) = c('dataset', 'learning_rate', 'acc_train', 'acc_test')
colnames(results_01) = c('dataset', 'learning_rate', 'acc_train', 'acc_test')
colnames(results_35) = c('dataset', 'learning_rate', 'acc_train', 'acc_test')
```

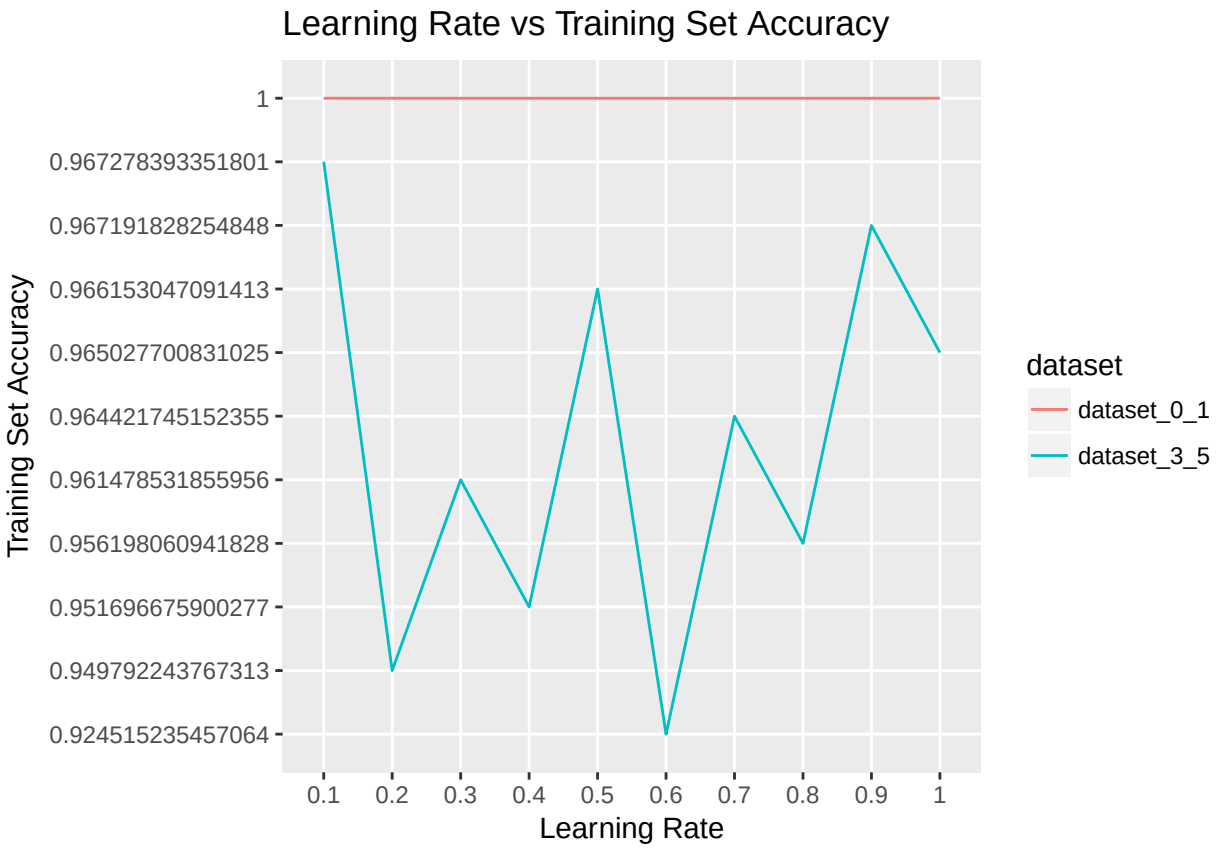
```
print(results_01)
```

```
##        dataset learning_rate acc_train          acc_test
## 1  dataset_0_1           0.1         1 0.999527186761229
## 2  dataset_0_1           0.2         1 0.999527186761229
## 3  dataset_0_1           0.3         1 0.999054373522459
## 4  dataset_0_1           0.4         1 0.999054373522459
## 5  dataset_0_1           0.5         1 0.999054373522459
## 6  dataset_0_1           0.6         1 0.998581560283688
## 7  dataset_0_1           0.7         1                 1
## 8  dataset_0_1           0.8         1 0.999054373522459
## 9  dataset_0_1           0.9         1 0.999054373522459
## 10 dataset_0_1             1         1 0.999054373522459
```

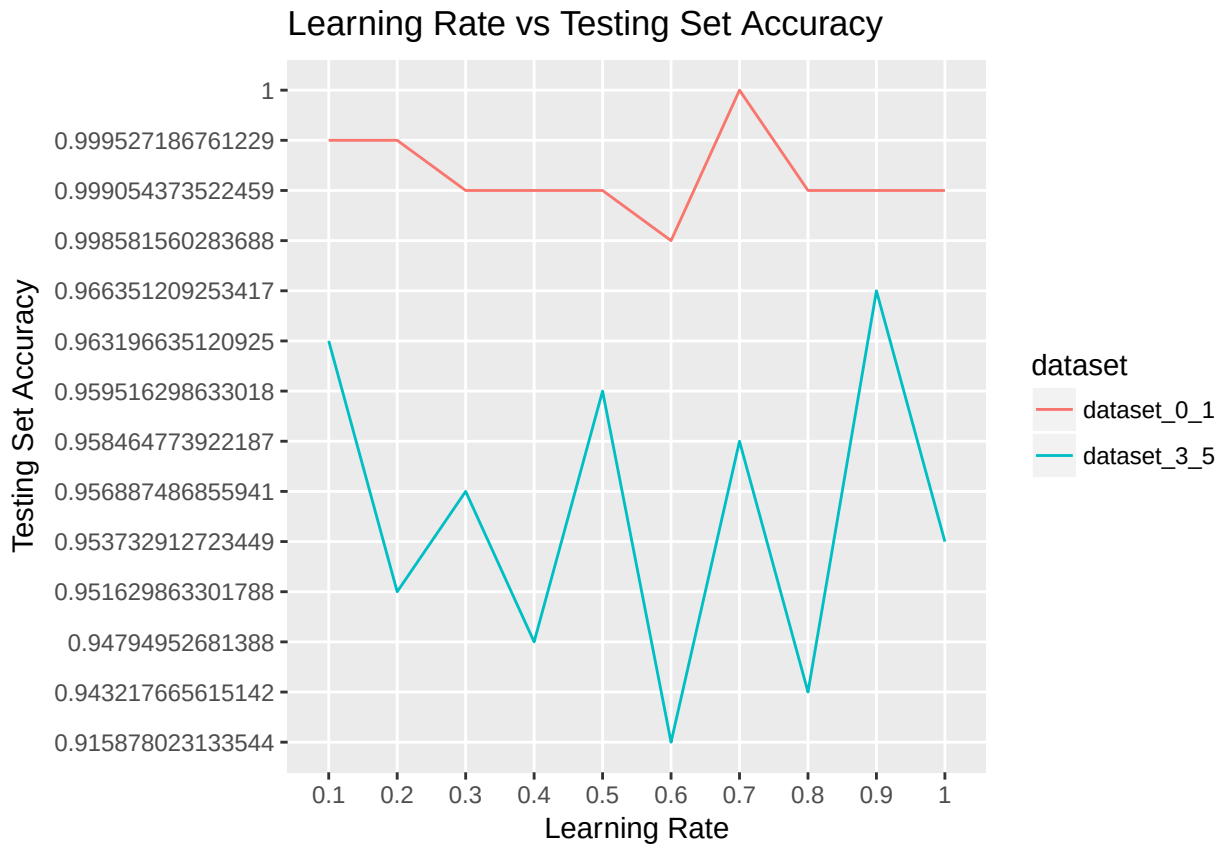
```
print(results_35)
```

```
##        dataset learning_rate         acc_train          acc_test
## 1  dataset_3_5           0.1 0.967278393351801 0.963196635120925
## 2  dataset_3_5           0.2 0.949792243767313 0.951629863301788
## 3  dataset_3_5           0.3 0.961478531855956 0.956887486855941
## 4  dataset_3_5           0.4 0.951696675900277  0.94794952681388
## 5  dataset_3_5           0.5 0.966153047091413 0.959516298633018
## 6  dataset_3_5           0.6 0.924515235457064 0.915878023133544
## 7  dataset_3_5           0.7 0.964421745152355 0.958464773922187
## 8  dataset_3_5           0.8 0.956198060941828 0.943217665615142
## 9  dataset_3_5           0.9 0.967191828254848 0.966351209253417
## 10 dataset_3_5             1 0.965027700831025 0.953732912723449
```

```
ggplot(data=results_df, aes(x=learning_rate, y=acc_train, group=dataset, color=dataset)) +
  geom_line() +
  ggtitle("Learning Rate vs Training Set Accuracy") +
  xlab("Learning Rate") +
  ylab("Training Set Accuracy")
```




```
ggplot(data=results_df, aes(x=learning_rate, y=acc_test, group=dataset, color=dataset)) +
  geom_line() +
  ggtitle("Learning Rate vs Testing Set Accuracy") +
  xlab("Learning Rate") +
  ylab("Testing Set Accuracy")
```



By analyzing the learning rate and accuracy relationships above, it looks like they are relatively consistent when it comes to comparing to training and test sets, as there are two distinct patterns for the sets 0/1 and 3/5 respectively. For the data set 0/1, the accuracy is relatively stable at 1 for all learning rates. However, there seems to be two maximum accuracies reached when the learning rates are either 0.4 or 0.9, with 0.4 being the higher accuracy of the two. This is true for both 0/1 and 3/5 in the test sets and for 3/5 in the training set, as the accuracy for 0/1 in the trainign set is all 100%. Therefore, from the observation, we can conclude the accuracy is at the highest when we have a learning rate of about 0.4.

From the analysis above, we do see that the data set for 0/1 for both training and testing have higher accuracies. This could first be explained by the fact that we have more training sets available for the 0/1 data set. We have 12665 samples for 0/1, whereas 11552 samples for 3/5, this allows more training to be done. In addition, the data set for 3/5 seems to be harder to predict, this might be due to the fact the features we are using are not as accurate to represent these images. Hence, we have lower training set accuracies, which further lead to lower testing accuracies.

This assignment deals with binary classification only. However, for a multiclass classification, we could actually decouple the problem into n distinct binary classification problems. For example, if we are given a dataset with 0/1/3/5 all together. We could decouple this into the binary classification problems 0 vs 1/3/5, 1 vs 0/3/5, 3 vs 0/1/5, etc. This allows us to apply the same concepts of stochastic gradient descent to these binary classification problems. We will be predicting one variable against all the others bunched together. For example, if we are predicting 0 vs 1/3/5, if the probability of 0 is higher, than we know the prediction is 0. However, is 1/3/5 is higher, we can then predict from 1/3/5 with 1 vs 3/5, 3 vs 1/5, etc and so on.

For the training sets, the maximum accuracy for data set 0/1 is at 1 at all learning rates, for data set 3/5 is at 0.962171052631579 at learning rate 0.4. For the testing sets, the maximum accuracy for data set 0/1 is at 0.999527186761229 at learning rates 0.2, 0.4 and 0.9, for testing data set 3/5 is at 0.956887486855941 at learning rate 0.4.

3. Learning Curves [30 points]

```
data_percentage = seq(0.1, 1.0, by = 0.1)
rate = 0.4
curves_01 = list()
curves_35 = list()

for (i in 1:length(data_percentage)) {
  percent = data_percentage[i]
  shuffled_01 = bind_and_shuffle(train_data_0_1, train_labels_0_1)
  shuffled_35 = bind_and_shuffle(train_data_3_5, train_labels_3_5)
  sample_01 = shuffled_01[sample(nrow(shuffled_01), percent*nrow(train_data_0_1)), ]
  sample_35 = shuffled_35[sample(nrow(shuffled_35), percent*nrow(train_data_3_5)), ]

  data_01 = sample_01[, -786]
  labels_01 = as.matrix(sample_01[, 786])

  data_35 = sample_35[, -786]
  labels_35 = as.matrix(sample_35[, 786])

  acc_train_01 = vector(mode="numeric")
  acc_test_01 = vector(mode="numeric")

  acc_train_35 = vector(mode="numeric")
  acc_test_35 = vector(mode="numeric")

  for (j in 1:5) {
    exp_01 = model(data_01, labels_01, test_data_0_1, test_labels_0_1, rate)
    exp_35 = model(data_35, labels_35, test_data_3_5, test_labels_3_5, rate)

    acc_train_01 = append(acc_train_01, exp_01[[2]])
    acc_test_01 = append(acc_test_01, exp_01[[3]])

    acc_train_35 = append(acc_train_35, exp_35[[2]])
    acc_test_35 = append(acc_test_35, exp_35[[3]])
  }

  curves_01[[i]] = c('dataset_0_1', percent, mean(acc_train_01), mean(acc_test_01))
  curves_35[[i]] = c('dataset_3_5', percent, mean(acc_train_35), mean(acc_test_35))
}

total_curves = append(curves_01, curves_35)
curves_01 = as.data.frame(do.call(rbind, curves_01))
curves_35 = as.data.frame(do.call(rbind, curves_35))
curves_df = as.data.frame(do.call(rbind, total_curves))
colnames(curves_df) = c('dataset', 'data_percentage', 'acc_train', 'acc_test')
colnames(curves_01) = c('dataset', 'data_percentage', 'acc_train', 'acc_test')
colnames(curves_35) = c('dataset', 'data_percentage', 'acc_train', 'acc_test')
```

```
print(curves_01)
```

```
##        dataset data_percentage acc_train          acc_test
## 1  dataset_0_1             0.1         1 0.998770685579196
## 2  dataset_0_1             0.2         1 0.999621749408983
```

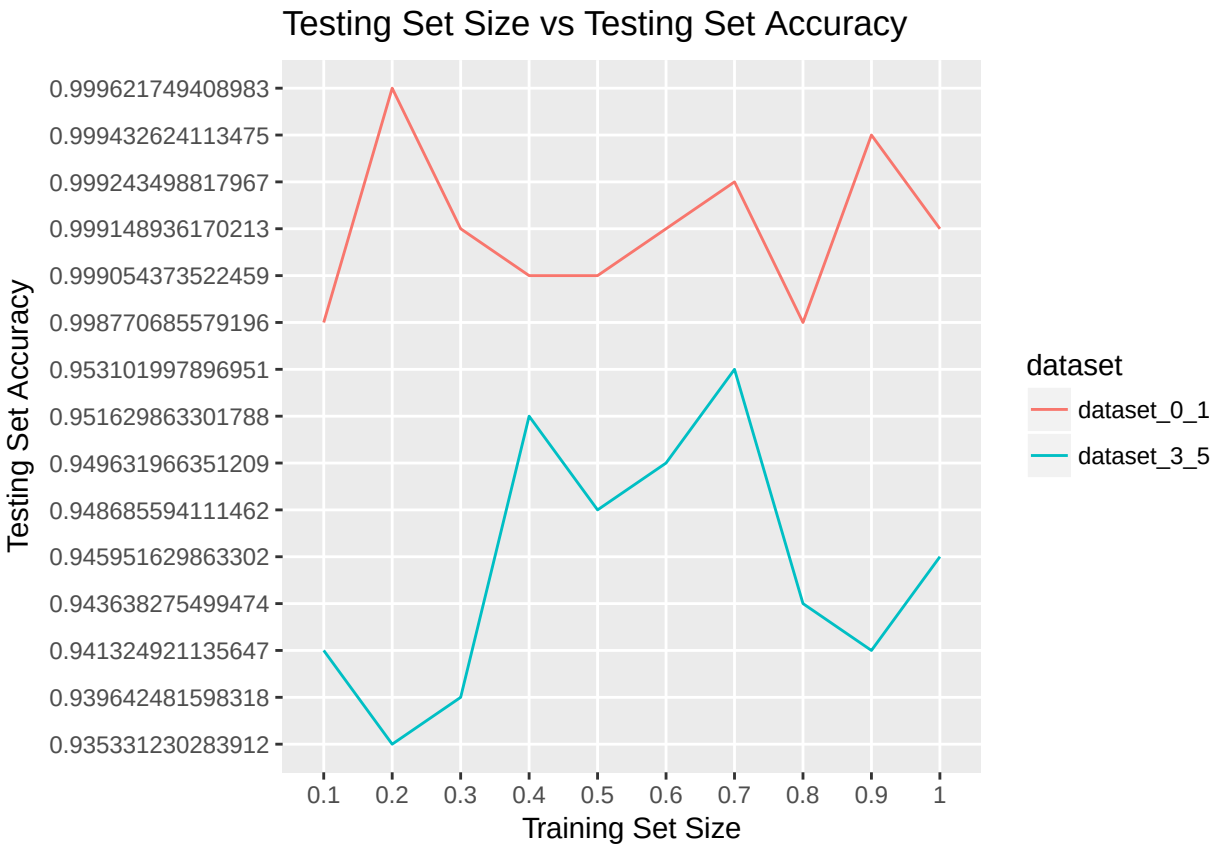
```
## 3  dataset_0_1             0.3         1 0.999148936170213
## 4  dataset_0_1             0.4         1 0.999054373522459
## 5  dataset_0_1             0.5         1 0.999054373522459
## 6  dataset_0_1             0.6         1 0.999148936170213
## 7  dataset_0_1             0.7         1 0.999243498817967
## 8  dataset_0_1             0.8         1 0.998770685579196
## 9  dataset_0_1             0.9         1 0.999432624113475
## 10 dataset_0_1               1         1 0.999148936170213
```

```
print(curves_35)
```

```
##        dataset data_percentage         acc_train          acc_test
## 1  dataset_3_5             0.1 0.991341991341991 0.941324921135647
## 2  dataset_3_5             0.2 0.964242424242424 0.935331230283912
## 3  dataset_3_5             0.3 0.958499278499279 0.939642481598318
## 4  dataset_3_5             0.4 0.966926406926407 0.951629863301788
## 5  dataset_3_5             0.5 0.967070637119114 0.948685594111462
## 6  dataset_3_5             0.6 0.960727167796855 0.949631966351209
## 7  dataset_3_5             0.7 0.966312144447193 0.953101997896951
## 8  dataset_3_5             0.8 0.955242939075858 0.943638275499474
## 9  dataset_3_5             0.9  0.95021161985379 0.941324921135647
## 10 dataset_3_5               1 0.953618421052632 0.945951629863302
```

```
ggplot(data=curves_df, aes(x=data_percentage, y=acc_train, group=dataset, color=dataset)) +
  geom_line() +
  ggtitle("Traing Set Size vs Training Set Accuracy") +
  xlab("Traing Set Size") +
  ylab("Training Set Accuracy")
```


```
ggplot(data=curves_df, aes(x=data_percentage, y=acc_test, group=dataset, color=dataset)) +
  geom_line() +
  ggtitle("Testing Set Size vs Testing Set Accuracy") +
  xlab("Training Set Size") +
  ylab("Testing Set Accuracy")
```



From the learning curves, there is a general pattern that as we increase the training set size, the accuracy for the testing set increases. This makes sense as when we have more traning samples, our model is more well trained and hence will do better on the testing sets.