Homework 3

Code

This is an R Markdown (http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

Hide

```
mnist_train=read.csv("mnist_train.csv", header=FALSE)
mnist_test=read.csv("mnist_test.csv", header=FALSE)
```

Hide

```
test_0_1=mnist_test[,mnist_test[nrow(mnist_test),]==0|mnist_test[nrow(mnist_test),]==
1]
test_3_5=mnist_test[,mnist_test[nrow(mnist_test),]==3|mnist_test[nrow(mnist_test),]==
5]
train_0_1=mnist_train[,mnist_train[nrow(mnist_train),]==0|mnist_train[nrow(mnist_trai
n),]==1]
train_3_5=mnist_train[,mnist_train[nrow(mnist_train),]==3|mnist_train[nrow(mnist_trai
n),]==5]
```

Hide

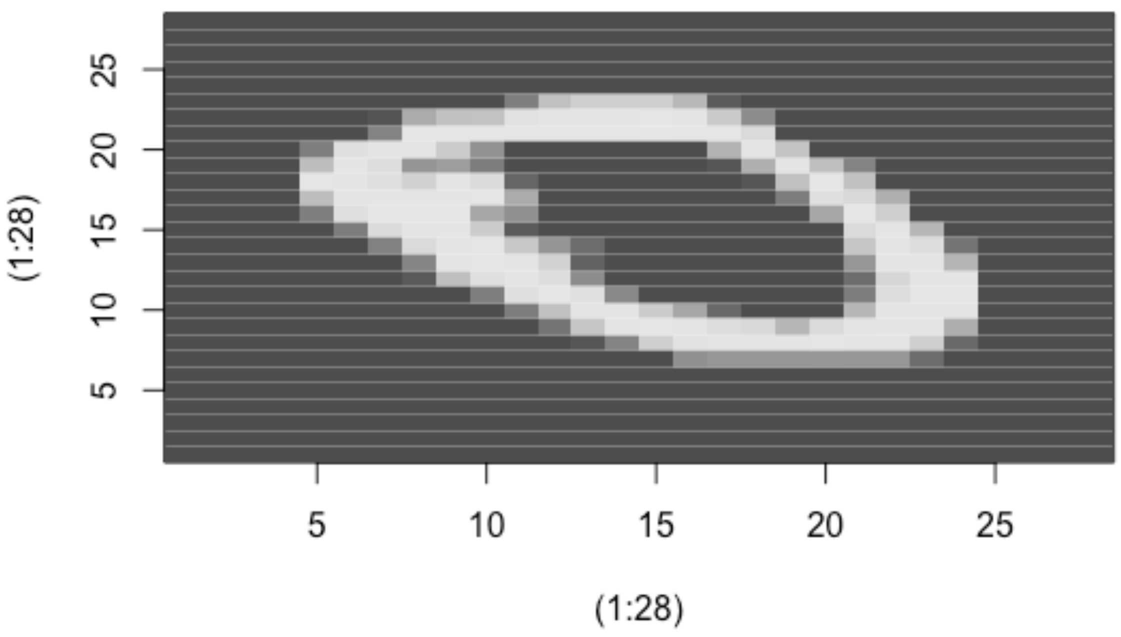
```
true_label_test_0_1=test_0_1[nrow(test_0_1),]
true_label_test_3_5=test_3_5[nrow(test_3_5),]
true_label_train_0_1=train_0_1[nrow(train_0_1),]
true_label_train_3_5=train_3_5[nrow(train_3_5),]
```

Hide

```
test_0_1=test_0_1[-nrow(test_0_1),]
test_3_5=test_3_5[-nrow(test_3_5),]
train_0_1=train_0_1[-nrow(train_0_1),]
train_3_5=train_3_5[-nrow(train_3_5),]
```

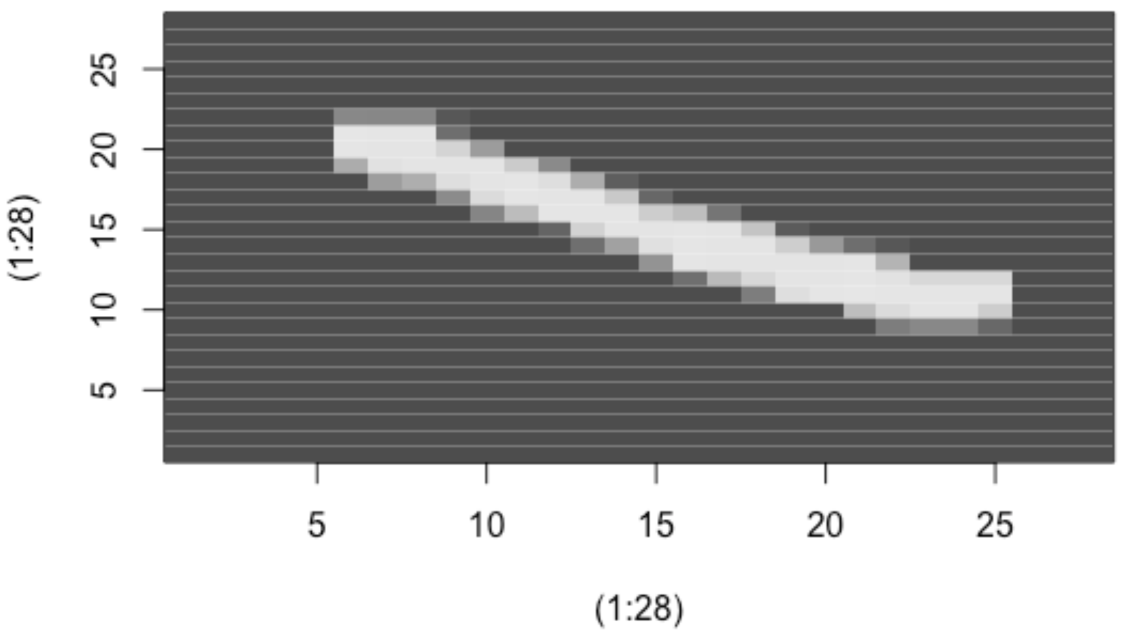
Hide

```
image((1:28), (1:28), matrix(data=as.numeric(train_0_1[,true_label_train_0_1[1,]==0][
,1]), nrow=sqrt(nrow(train_0_1)), ncol=sqrt(nrow(train_0_1))), col=gray.colors(256))
```



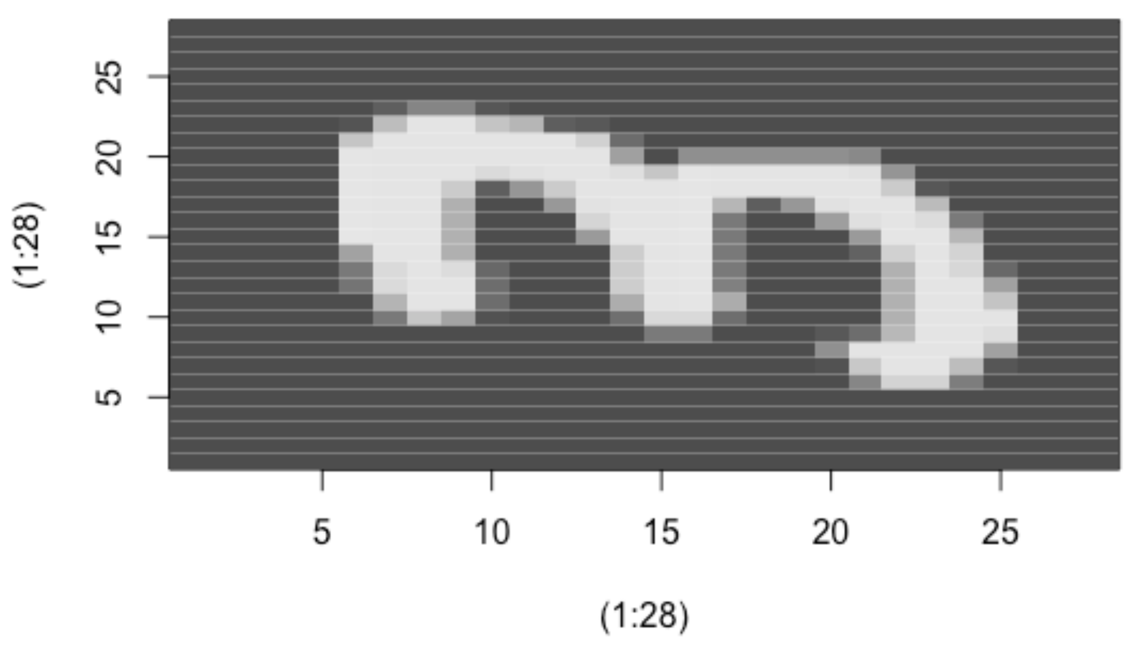
Hide

```
image((1:28), (1:28), matrix(data=as.numeric(train_0_1[,true_label_train_0_1[1,]==1][
,1]), nrow=sqrt(nrow(train_0_1)), ncol=sqrt(nrow(train_0_1))), col=gray.colors(256))
```



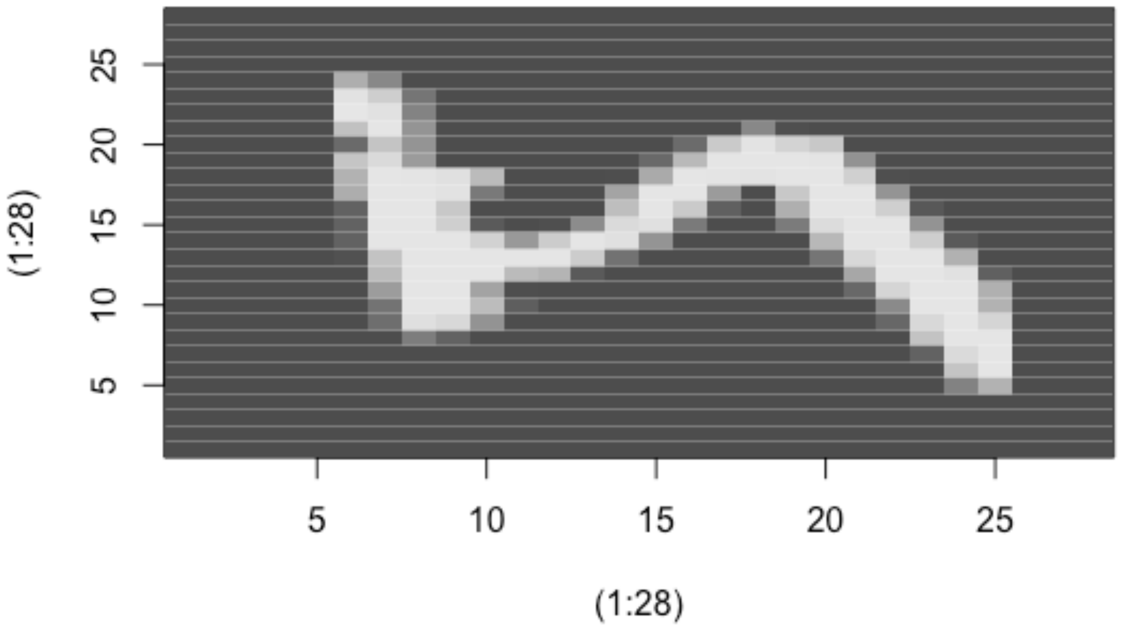
Hide

```
image((1:28), (1:28), matrix(data=as.numeric(train_3_5[,true_label_train_3_5[1,]==3][
,1]), nrow=sqrt(nrow(train_3_5)), ncol=sqrt(nrow(train_3_5))), col=gray.colors(256))
```



Hide

```
image((1:28), (1:28), matrix(data=as.numeric(train_3_5[,true_label_train_3_5[1,]==5][
,1]), nrow=sqrt(nrow(train_3_5)), ncol=sqrt(nrow(train_3_5))), col=gray.colors(256))
```



1. Theory

a. Write down the formula for computing the gradient of the loss function used in Logistic Regression. Specify what each variable represents in the equation.

Answer: Below is the formula for gradient descent process:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{1 + \exp(-\theta^\intercal x^i)} - y^i \right) x_j^i$$

In the formula: θ_j' is the updated value of the parameter θ_j for the cost function.

θ_j is one of the parameter in the parameter vector θ

θ is the parameter vector to be optmized in the cost function

α is the learning rate

m is the number of training data

x^i is one vector of the training data vectors

y^i is true class/label for the training data vector x^i

x_j^i is the x value for parameter θ_j

b. Write pseudocode for training a model using Logistic Regression.

Answer: Pseudocode for training a Logistic Regression model:

matrix_x=matrix(data=input_data, row_number=nrow(training_data), column_number=ncol(training_data))
matrix_y=matrix(data=labels, row_number=nrow(training_data), column_number=1)
matrix_theta=matrix(data=initial_theta, row_number=ncol(training_data), column_number=1)

gradient_function=function(matrix_x, matrix_y,matrix_theta){ theta=matrix_theta-
alpha*(1/nrow(training_data))transpose(matrix_x)%*%(1/(1+exp(-matrix_x%*%matrix_theta))-matrix_y)
return(theta) }

sigmoid_function=function(z){ g=1/(1+exp(-z)) return(g) }

cost_function=function(matrix_theta){ g=sigmoid_function(matrix_x%*%matrix_theta) cost_J=((transpose(-
matrix_y)%*%log(g))-(transpose(1-matrix_y)%*%log(1-g))) return(cost_J) }

while(J>threshold){ matrix_theta=gradient_function(matrix_x, matrix_y, matrix_theta)
J=cost_function(matrix_theta) }

c. Calculate the number of operations per gradient descent iteration. (Hint: Use variable n for number of examples and d for dimensionality.) Answer: In my implementation, I use matrix to calculate the gradient descent. For each iteration, the number of operation is 1. In calculation, the operation time is m*n.

2. Implementation Implement Logistic Regression using Gradient Descent. Notes: . You can use any gradient descent technique (batch or stochastic). . Choose any set of initial parameters for gradient descent. You will use the same for question 3. . Choose any convergence criteria. You will use the same for question 3. . Try to vectorize code by avoiding for loops.

Answer: In this implementation, I adopted Batch implementation. I used functions as model training, which can be repeated easily.

Hide

```
sigmoid=function(z){
  g=1/(1+exp(-z))
  return(g)
}
logit_model=function(input_data, class_data, theta_initial, alpha,epsilon ){

  if(length(unique(unlist(class_data)))==2){

    init=0
    for(i in unique(unlist(class_data)) ){
      if(!(i%in%c(0,1))) {
        class_data[class_data==3]=0
        class_data[class_data==5]=1

      }}
  }


  x=t(as.matrix(input_data))
  y=t(as.matrix(class_data))
  theta=matrix(data=rep(theta_initial, ncol(x)), nrow=ncol(x), ncol=1)
  J=epsilon+1
  while(J>epsilon)
  {

    theta=theta-alpha*(t(x)%*%(1/(1+exp(-x%*%theta))-y)/(nrow(x)))
    J=(t(-y)%*%log(sigmoid(x%*%theta))-t(1-y)%*%log(1-sigmoid(x%*%theta)))
    print(J)
  }
  return(theta)
}
```

Hide

```
model_test=function(theta, training_input_data, training_class_data, test_input_data,
test_class_data){

  if(length(unique(unlist(test_class_data)))==2){

    init=0
    for(i in unique(unlist(test_class_data)) ){
      if(!(i%in%c(0,1))) {
        test_class_data[test_class_data==3]=0
        test_class_data[test_class_data==5]=1

      }}
  }
  if(length(unique(unlist(training_class_data)))==2){

    init=0
    for(i in unique(unlist(training_class_data)) ){
      if(!(i%in%c(0,1))) {
        training_class_data[training_class_data==3]=0
        training_class_data[training_class_data==5]=1

      }}
  }

  prediction_train=round(sigmoid(t(as.matrix(training_input_data))%*%theta))
  error_train=sum(abs(prediction_train-t(as.matrix(training_class_data))))/nrow(t(as.
matrix(training_class_data)))
  prediction_test=round(sigmoid(t(as.matrix(test_input_data))%*%theta))
  error_test=sum(abs(prediction_test-t(as.matrix(test_class_data))))/nrow(t(as.matrix
(test_class_data)))

  J_training=(t(-t(as.matrix(training_class_data)))%*%log(sigmoid(t(as.matrix(trainin
g_input_data))%*%theta))-t(1-t(as.matrix(training_class_data)))%*%log(1-sigmoid(t(as.
matrix(training_input_data))%*%theta)))
  J_test=(t(-t(as.matrix(test_class_data)))%*%log(sigmoid(t(as.matrix(test_input_data
))%*%theta))-t(1-t(as.matrix(test_class_data)))%*%log(1-sigmoid(t(as.matrix(test_inpu
t_data))%*%theta)))

  print(cat("Accuracy of Training Set: ", 1-error_train))
  print(cat("Accuracy of Test Set: ", 1-error_test))
  return(c(1-error_train, 1-error_test,J_training,J_test))

}
```

3. Training Train and evaluate the code from question 2 on MNIST dataset.

Hide

```
sample_and_shuffle=function(data, sample_perc){
  new_data=sample(data, sample_perc*(ncol(data)), replace=FALSE)
  return(new_data)
}
```

Hide

```
repeat_modeling=function(training_data, test_data, class,repeat_time, size,size_step,
initial_theta, alpha, cost){
result=c()
for(i in c(1:repeat_time)){
spsf_train=sample_and_shuffle(training_data,size)
if(identical(unlist(class),unlist(c(3,5)))){
spsf_train=spsf_train[,spsf_train[nrow(spsf_train),]==3|spsf_train[nrow(spsf_train),]
==5]
true_label_spsf_train=spsf_train[nrow(spsf_train),]
spsf_train=spsf_train[-nrow(spsf_train),]
spsf_test=test_data
spsf_test=spsf_test[,spsf_test[nrow(spsf_test),]==3|spsf_test[nrow(spsf_test),]==5]
true_label_spsf_test=spsf_test[nrow(spsf_test),]
spsf_test=spsf_test[-nrow(spsf_test),]
}
else if(identical(unlist(class),unlist(c(0,1)))){
spsf_train=spsf_train[,spsf_train[nrow(spsf_train),]==0|spsf_train[nrow(spsf_train),]
==1]
true_label_spsf_train=spsf_train[nrow(spsf_train),]
spsf_train=spsf_train[-nrow(spsf_train),]
spsf_test=test_data
spsf_test=spsf_test[,spsf_test[nrow(spsf_test),]==0|spsf_test[nrow(spsf_test),]==1]
true_label_spsf_test=spsf_test[nrow(spsf_test),]
spsf_test=spsf_test[-nrow(spsf_test),]
} else{ print("Wrong Class")}
print(nrow(t(spsf_train)))
print(nrow(t(true_label_spsf_train)))
print(nrow(t(spsf_test)))
print(nrow(t(true_label_spsf_test)))
spsf_theta=logit_model(spsf_train,true_label_spsf_train, initial_theta,alpha, cost)
accuracy=model_test(spsf_theta, spsf_train,true_label_spsf_train, spsf_test, true_lab
el_spsf_test)
result=rbind(result,c(i, size, initial_theta, alpha, cost, accuracy[1],accuracy[2],ac
curacy[3],accuracy[4]))
size=size-size_step
}
return(result)
}
```

a. Train 2 models, one on the train_0_1 set and another on train_3_5, and report the training and test accuracies. Answer:

Hide

```
colnames(q_1a_result_01)=c("i", "size","initial_theta","alpha","cost","train_accuracy
", "test_accuracy",  "train_cost", "test_cost")
colnames(q_1a_result_35)=c("i", "size","initial_theta","alpha","cost","train_accuracy
", "test_accuracy",  "train_cost", "test_cost")
print(q_1a_result_01)
```

```
     i size initial_theta alpha cost train_accuracy test_accuracy
[1,] 1    1           0.1   0.8  200      0.9958942     0.9981087
     train_cost test_cost
[1,]   197.8829  17.09168
```

Hide

```
print(q_1a_result_35)
```

```
     i size initial_theta alpha cost train_accuracy test_accuracy
[1,] 1    1           0.1   0.8 2000      0.9313539     0.9416404
     train_cost test_cost
[1,]   1979.204  269.3895
```

b. Repeat 3a 10 times, i.e. you should obtain 10 train and test accuracies for each set.

Calculate the average train and test accuracies over the 10 runs, and report them.

Hide

```
df_result_01=as.data.frame(q_1b_result_01)
df_result_35=as.data.frame(q_1b_result_35)
colnames(df_result_01)=c("i", "size","initial_theta","alpha","cost","train_accuracy",
"test_accuracy",  "train_cost", "test_cost")
colnames(df_result_35)=c("i", "size","initial_theta","alpha","cost","train_accuracy",
"test_accuracy",  "train_cost", "test_cost")
print(q_1b_result_01)
```

```
      i size initial_theta alpha cost train_accuracy test_accuracy
 [1,]  1  1.0           0.1   0.8  200      0.9958942     0.9981087
 [2,]  2  0.9           0.1   0.8  200      0.9950912     0.9976359
 [3,]  3  0.8           0.1   0.8  200      0.9945914     0.9976359
 [4,]  4  0.7           0.1   0.8  200      0.9928757     0.9976359
 [5,]  5  0.6           0.1   0.8  200      0.9931937     0.9976359
 [6,]  6  0.5           0.1   0.8  200      0.9898862     0.9966903
 [7,]  7  0.4           0.1   0.8  200      0.9891620     0.9938534
 [8,]  8  0.3           0.1   0.8  200      0.9899895     0.9938534
 [9,]  9  0.2           0.1   0.8  200      0.9795678     0.9881797
[10,] 10  0.1           0.1   0.8  200      0.9857482     0.9891253
      train_cost test_cost
 [1,]  197.88290  17.09168
 [2,]  192.62222  18.07694
 [3,]  190.08484  20.02764
 [4,]  194.32242  22.78142
 [5,]  163.38470  23.19174
 [6,]  177.53379  27.16901
 [7,]  169.57567  38.48396
 [8,]  132.25069  39.75660
 [9,]  150.33895  77.54499
[10,]   61.22786  72.83059
```

Hide

```
print(q_1b_result_35)
```

```
      i size initial_theta alpha cost train_accuracy test_accuracy
 [1,]  1  1.0           0.1   0.8 2000      0.9313539     0.9416404
 [2,]  2  0.9           0.1   0.8 2000      0.9209387     0.9348055
 [3,]  3  0.8           0.1   0.8 2000      0.9093071     0.9232387
 [4,]  4  0.7           0.1   0.8 2000      0.9100542     0.9269190
 [5,]  5  0.6           0.1   0.8 2000      0.9038846     0.9195584
 [6,]  6  0.5           0.1   0.8 2000      0.8620629     0.8690852
 [7,]  7  0.4           0.1   0.8 2000      0.8638008     0.8732913
 [8,]  8  0.3           0.1   0.8 2000      0.8392550     0.8491062
 [9,]  9  0.2           0.1   0.8 2000      0.8432709     0.8512093
[10,] 10  0.1           0.1   0.8 2000      0.5108601     0.5315457
     train_cost test_cost
 [1,]   1979.204  269.3895
 [2,]   1919.909  296.1596
 [3,]   1906.035  346.4500
 [4,]   1901.876  375.4466
 [5,]   1730.010  395.8560
 [6,]   1820.619  560.9905
 [7,]   1767.071  655.1655
 [8,]   1576.072  811.4105
 [9,]   1013.446  787.5390
[10,]   1599.077 2481.7610
```

Hide

```
(aggregate(df_result_01,list(df_result_01$initial_theta),mean))[,c("train_accuracy",
"test_accuracy")]
```

train_accuracy <dbl>	test_accuracy <dbl>
0.9906	0.9950355

1 row

Hide

```
(aggregate(df_result_35,list(df_result_35$initial_theta),mean))[,c("train_accuracy",
"test_accuracy")]
```

train_accuracy <dbl>	test_accuracy <dbl>
0.8494788	0.86204

1 row

c. For 0,1 and 3,5 cases, explain if you observe any difference you in accuracy. Also, explain why do you think this difference might be.

Answer: The accuracy in 1/0 data set is much higher than 3/5 dataset. It might be for 3/5, the epsilon is much bigger(because the dataset is not consistent as 0/1 dataset is). It is much harder to converge.

d. This assignment deals with binary classification. Explain what you would do if you had more than two classes to classify, using logistic regression.

Answer: For multi-class classification, we can treat multiple classes as two classes recursively. For each step, divide dataset into two classes, one is a single class and another is the combination of rest classes, then implement binary classification to this dataset. Repeat this process and we can classify all of single classes.

4. Evaluation In this question, you will experiment with different sets of parameters and observe how your model performs. This should be done ONLY for 3,5 classification.

Hide

```
parameter_eval=function(training_data,test_data,class,repeat_time,experiment_time,siz
e,size_step,initial_theta,theta_step,alpha,convergence_cost,convergence_cost_step){

  if(theta_step!=0){
    theta=initial_theta
    result=c()
    experiment_result=c()
    for(i in c(1:experiment_time)){
      result=repeat_modeling(training_data,test_data, class,repeat_time,size,size_ste
p,theta,alpha,convergence_cost)
      result=cbind(result,rep(theta,repeat_time))
      experiment_result=rbind(experiment_result,result)
      theta=theta-theta_step
    }
  }

    if(convergence_cost_step!=0){
    cost=convergence_cost
    result=c()
    experiment_result=c()
    for(i in c(1:experiment_time)){
      result=repeat_modeling(training_data,test_data, class,repeat_time,size,size_ste
p,initial_theta,alpha,cost)
      result=cbind(result,rep(cost,repeat_time))
      experiment_result=rbind(experiment_result,result)
      cost=cost-convergence_cost_step
    }
  }

  return(experiment_result)
}
```

a. Experiment with different initializations of the parameter used for gradient descent . Clearly mention the initial values of the parameter tried, run the same experiment as 3b using this initialization, report the average test and train accuracies obtained by using this initialization, mention which is set of initializations is the better.

Answer: I choose theta from 0.1 to 0.01, theta redcue 0.01 for every time. The result indicates the smaller theta is, the better the accuracy I get. In this case, the best theta is 0.01.

Hide

```
df_theta_eval_01=as.data.frame(theta_eval_01)
df_theta_eval_35=as.data.frame(theta_eval_35)
colnames(df_theta_eval_01)=c("i","size","initial_theta","alpha","cost","train_accurac
y", "test_accuracy", "train_cost", "test_cost","theta")
colnames(df_theta_eval_35)=c("i","size","initial_theta","alpha","cost","train_accurac
y", "test_accuracy",  "train_cost", "test_cost","theta")
agg_theta_eval_01=(aggregate(df_theta_eval_01,list(df_theta_eval_01$theta),mean))[,c(
"theta","train_accuracy", "test_accuracy")]
agg_theta_eval_35=(aggregate(df_theta_eval_35,list(df_theta_eval_35$theta),mean))[,c(
"theta","train_accuracy", "test_accuracy")]
```

Hide

```
print(agg_theta_eval_01)
```

theta <dbl>	train_accuracy <dbl>	test_accuracy <dbl>
0.01	0.9950881	0.9972104
0.02	0.9951240	0.9974468
0.03	0.9948820	0.9977305
0.04	0.9950549	0.9976359
0.05	0.9949257	0.9976359
0.06	0.9932566	0.9974468
0.07	0.9932264	0.9969740
0.08	0.9923223	0.9957920
0.09	0.9900406	0.9954137
0.10	0.9904015	0.9955083

1-10 of 10 rows

Hide

```
print(agg_theta_eval_35)
```

theta <dbl>	train_accuracy <dbl>	test_accuracy <dbl>
0.01	0.8351063	0.8410095

0.02	0.8651051	0.8732387
0.03	0.8890859	0.8974238
0.04	0.8859272	0.8917981
0.05	0.8762673	0.8875920
0.06	0.8415882	0.8484227
0.07	0.8537177	0.8607256
0.08	0.8469667	0.8537855
0.09	0.8645834	0.8733438
0.10	0.8797420	0.8905889

1-10 of 10 rows

b. Experiment with different convergence criteria for gradient descent. Clearly mention the new criteria tried, run the same experiment as 3b using this new criteria, report average test and train accuracies obtained using this criteria, mention which set of criteria is better.

Answer: I choose cost/epsilon from 2100 to 2010, theta redcue 10 for every time. The result indicates the smaller cost/epsilon is, the better the accuracy I get. In this case, the best epsilon is 2030.

Hide

```
df_convergence_eval_01=as.data.frame(convergence_eval_01)
df_convergence_eval_35=as.data.frame(convergence_eval_35)
colnames(df_convergence_eval_01)=c("i","size","initial_theta","alpha","cost","train_a
ccuracy", "test_accuracy", "train_cost", "test_cost","cost")
colnames(df_convergence_eval_35)=c("i","size","initial_theta","alpha","cost","train_a
ccuracy", "test_accuracy",  "train_cost", "test_cost","cost")
agg_convergence_eval_01=(aggregate(df_convergence_eval_01,list(df_convergence_eval_01
$cost),mean))[,c("cost","train_accuracy", "test_accuracy")]
agg_convergence_eval_35=(aggregate(df_convergence_eval_35,list(df_convergence_eval_35
$cost),mean))[,c("cost","train_accuracy", "test_accuracy")]
```

Hide

```
print(agg_convergence_eval_01)
```

cost <dbl>	train_accuracy <dbl>	test_accuracy <dbl>
210	0.9891826	0.9945154
220	0.9886470	0.9943735

230	0.9870474	0.9933333
240	0.9887169	0.9945626
250	0.9866588	0.9940898
260	0.9881728	0.9939480
270	0.9880351	0.9940426
280	0.9863504	0.9932388
290	0.9862632	0.9926714
300	0.9862697	0.9927187

1-10 of 10 rows

Hide

```
print(agg_convergence_eval_35)
```

cost <dbl>	train_accuracy <dbl>	test_accuracy <dbl>
2010	0.8665077	0.8756046
2020	0.8741181	0.8896951
2030	0.8839609	0.8964774
2040	0.8792751	0.8909043
2050	0.8671875	0.8801262
2060	0.8781223	0.8860673
2070	0.8785395	0.8885910
2080	0.8770554	0.8871714
2090	0.8728617	0.8838591
2100	0.8748087	0.8850683

1-10 of 10 rows

5. Learning Curves In this question, you will experiment with different training set sizes and see how the accuracy changes with them. This question should be done for both 0,1 and 3,5 classification.

Hide

```
learning_curves=function(training_data,test_data,class,repeat_time,experiment_time,si
ze,size_step,initial_theta,alpha,convergence_cost){
  result=c()
  experiment_result=c()
  for(i in c(1:experiment_time)){
    result=repeat_modeling(training_data, test_data,class,repeat_time,size, 0, initia
l_theta, alpha, convergence_cost)
    result=cbind(result,rep(size,repeat_time))
    experiment_result=rbind(experiment_result,result)
    size=size-size_step
  }
  return(experiment_result)
}
```

a. For each set of classes (0,1 and 3,5), choose the following sizes to train on: 5%, 10%, 15% … 100% (i.e. 20 training set sizes). For each training set size, sample that many inputs from the respective complete training set (i.e. train_0_1 or train_3_5). Train your model on each subset selected, test it on the corresponding test set (i.e. test_0_1 or test_3_5), and graph the training and test set accuracy over each split (you should end up with TWO graphs - one showing training & test accuracy for 0,1 and another for 3,5 set). Remember to average the accuracy over 10 different divisions of the data each of the above sizes so the graphs will be less noisy. Comment on the trends of accuracy values you observe for each set.

Answer: In this question, I run 10 times for each size and reduce 5% to get a new training data size.

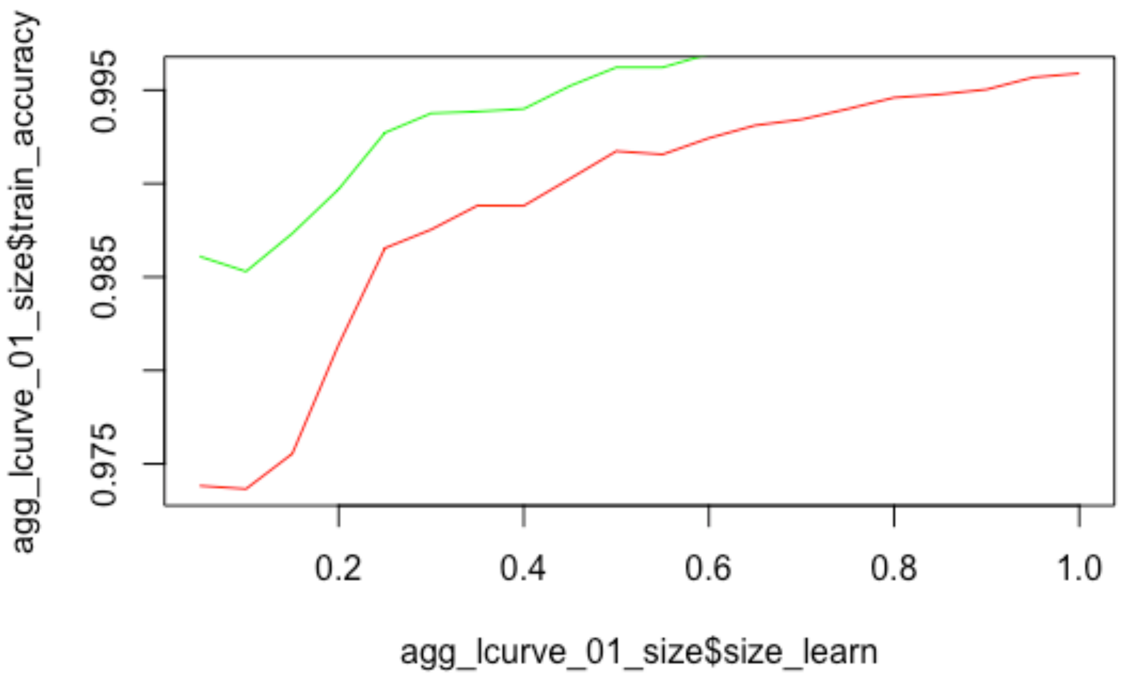
In the line plot, red line is the accuracy/loss for training data and the green line is the accuracy/loss for test data.

Hide

```
df_lcurve_01=as.data.frame(lcurve_01)
df_lcurve_35=as.data.frame(lcurve_35)
colnames(df_lcurve_01)=c("i","size","initial_theta","alpha","cost","train_accuracy",
"test_accuracy", "train_cost", "test_cost","size_learn")
colnames(df_lcurve_35)=c("i","size","initial_theta","alpha","cost","train_accuracy",
"test_accuracy", "train_cost", "test_cost","size_learn")
agg_lcurve_01_size=(aggregate(df_lcurve_01,list(df_lcurve_01$size_learn),mean))[,c("s
ize_learn","train_accuracy", "test_accuracy")]
agg_lcurve_35_size=(aggregate(df_lcurve_35,list(df_lcurve_35$size_learn),mean))[,c("s
ize_learn","train_accuracy", "test_accuracy")]
```

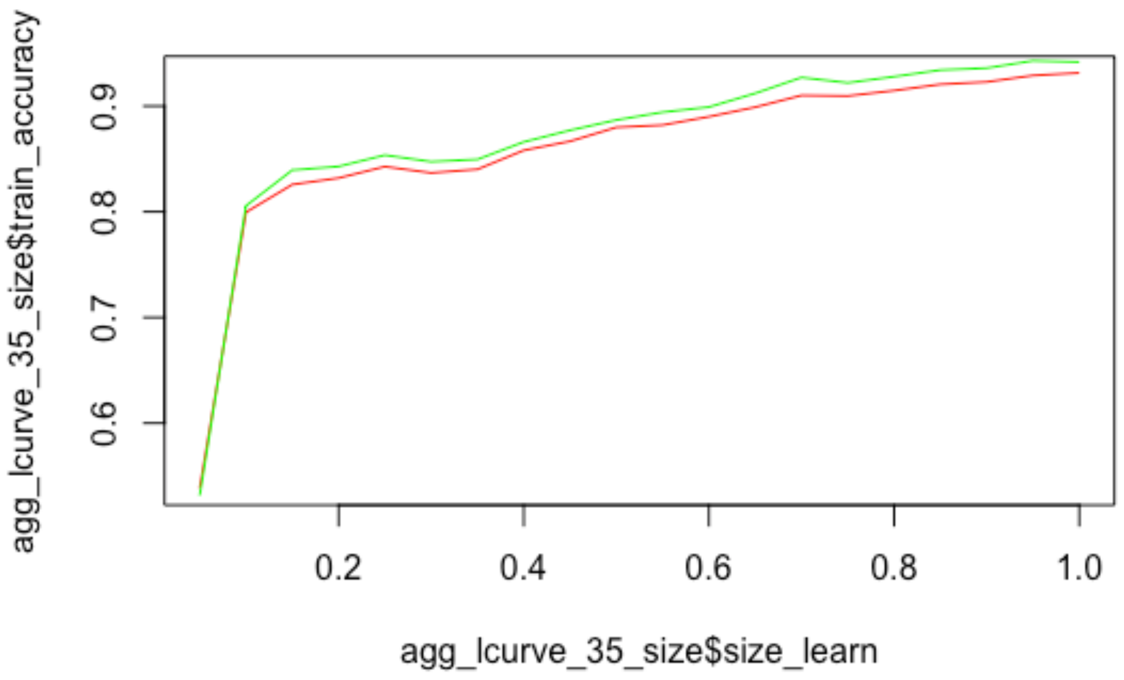
Hide

```
plot(agg_lcurve_01_size$size_learn,agg_lcurve_01_size$train_accuracy,type = "l",col="
red")
lines(agg_lcurve_01_size$size_learn,agg_lcurve_01_size$test_accuracy,col="green")
```



Hide

```
plot(agg_lcurve_35_size$size_learn,agg_lcurve_35_size$train_accuracy,type = "l",col="
red")
lines(agg_lcurve_35_size$size_learn,agg_lcurve_35_size$test_accuracy,col="green")
```



b. Repeat 5a, but instead of plotting accuracies, plot the logistic loss/negative log likelihood when training and testing, for each size. Comment on the trends of loss values you observe for each set.

Hide

```
agg_lcurve_01_cost=(aggregate(df_lcurve_01,list(df_lcurve_01$size_learn),mean))[,c("s
ize_learn","train_cost", "test_cost")]
agg_lcurve_35_cost=(aggregate(df_lcurve_35,list(df_lcurve_35$size_learn),mean))[,c("s
ize_learn","train_cost", "test_cost")]
```

Hide

```
print(agg_lcurve_01_cost)
```

size_learn <dbl>	train_cost <dbl>	test_cost <dbl>
0.05	50.74943	88.22508
0.10	101.92758	94.60069
0.15	139.79002	81.97481
0.20	140.82920	65.28317
0.25	132.25348	46.25264
0.30	146.59911	38.76930
0.35	161.19251	38.75359
0.40	176.90960	38.48928
0.45	175.91264	33.17141
0.50	164.43240	28.37762

1-10 of 20 rows Previous **1** 2 Next

Hide

```
print(agg_lcurve_35_cost)
```

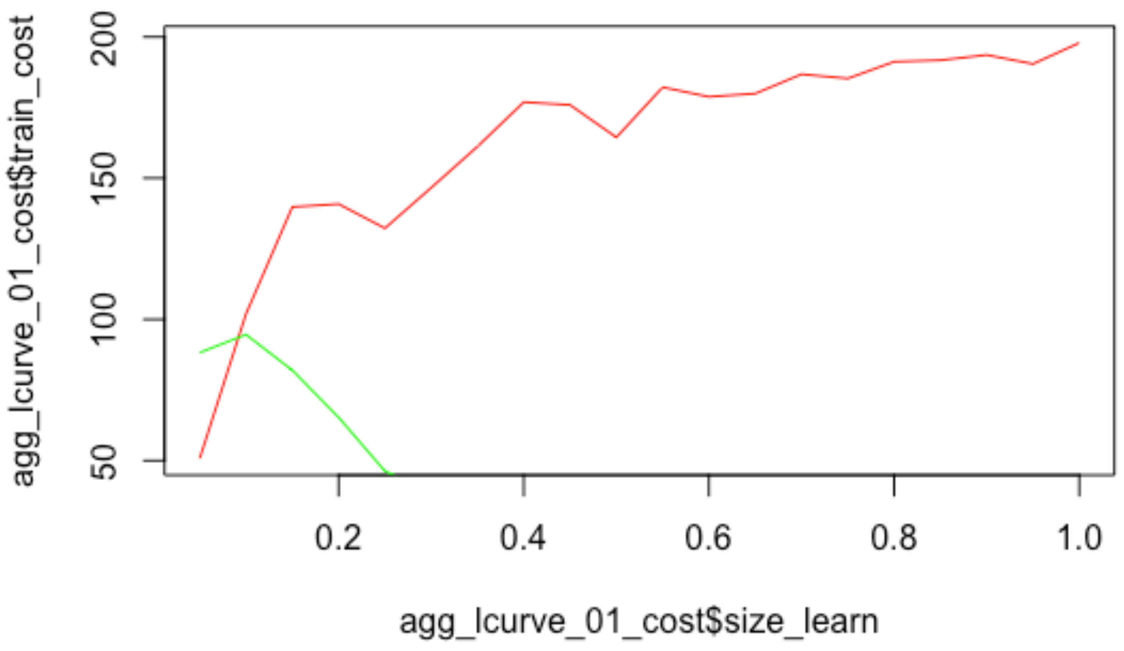
size_learn <dbl>	train_cost <dbl>	test_cost <dbl>
0.05	1810.1024	6212.0757
0.10	682.6093	1063.7399
0.15	837.2990	857.4729

0.20	1090.4104	830.4899
0.25	1243.1089	749.0089
0.30	1592.8234	798.3260
0.35	1778.1195	772.1190
0.40	1789.8019	679.7370
0.45	1816.6962	600.5987
0.50	1791.8436	540.8008

1-10 of 20 rows Previous **1** 2 Next

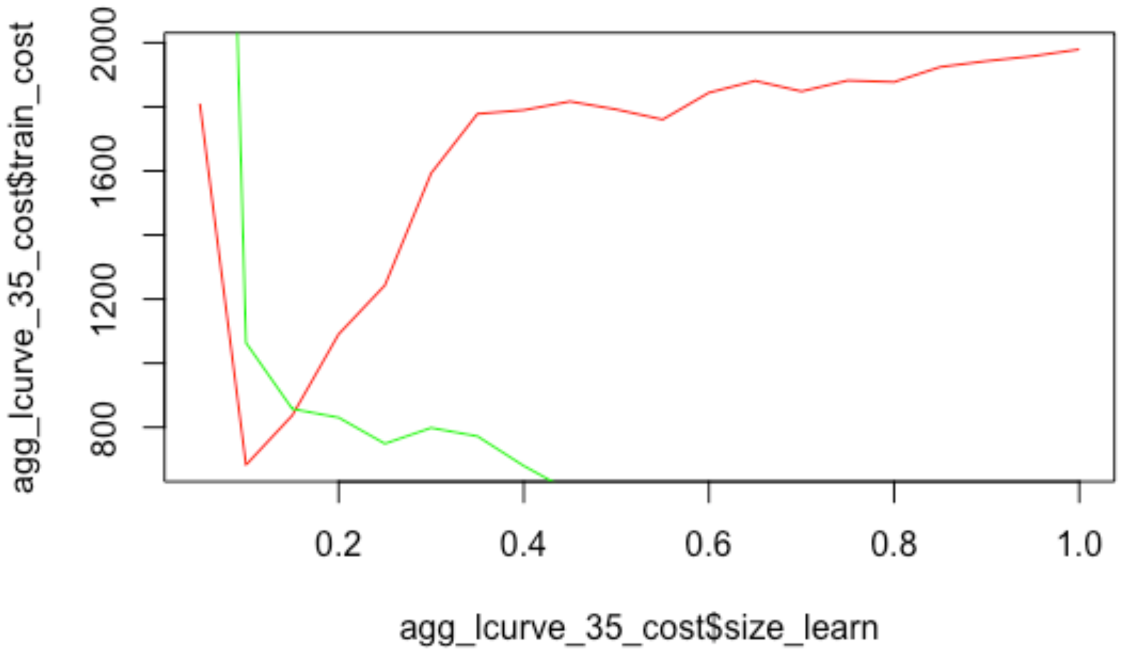
Hide

```
plot(agg_lcurve_01_cost$size_learn,agg_lcurve_01_cost$train_cost,type = "l",col="red"
)
lines(agg_lcurve_01_cost$size_learn,agg_lcurve_01_cost$test_cost,col="green")
```



Hide

```
plot(agg_lcurve_35_cost$size_learn,agg_lcurve_35_cost$train_cost,type = "l",col="red"
)
lines(agg_lcurve_35_cost$size_learn,agg_lcurve_35_cost$test_cost,col="green")
```



Notes: . You can choose to select the % of training data in any manner. If you choose random divisions, make sure you extract the correct corresponding labels for each image. . For each training set size (5%, 10%, etc.), you should run your model 10 times and average the accuracy. If 10 times is too computationally intensive, you can run 5 times. Clearly specify the number of times you have averaged on.