hw3_514

Homework 3 Problem Statement

Implement gradient descent for a logistic model using the squared error (SE) and the negative log likelihood (NLL) cost functions. The most important functions in your solution are

- b.prop: Computes the gradient of the cost function wrt to adjustable parameters b and w. Your solution should be "vectorized" and use vector/matrix operations to compute the gradient for the entire dataset on each call.
- f.prop: Computes model outputs. Your solution should be "vectorized"
- log.fit: Implements a simple gradient descent loop given fixed learning rate and the number of iterations
- cost: Compute cost for the entire data set

A good reference on logistic regression algorithms is Andre Ng's lecture notes on logistic regression

Step 1 Generate data

Source the provided hw3_514.R file. Use the function generate.1d.dataset to create a dataset. This function creates a 1 dimensional gaussian mixture dataset. A few plots are provided to visualize the data.

```
source('./hw3_514_solution.R')
```

```
## Loading required package: zeallot
```

```
## Loading required package: plyr
```

```
## Loading required package: mvtnorm
```

```
set.seed(1234)
require(MASS)
```

```
## Loading required package: MASS
```

```
m1=50;mu1=1;s1=.5
m2=50;mu2=2;s2=.7
c(xin,y) %<-% generate.1d.dataset(m1,mu1,s1,m2,mu2,s2)
plot(xin,col=y+3)
bayes.cut2=function(mu1,s1,mu2,s2){
  ff=function(x){dnorm(x,mu2,s2) - dnorm(x,mu1,s1)}
  uniroot(ff,c(mu1,mu2))$root
}

cut=bayes.cut2(mu1,s1,mu2,s2)
abline(h=cut)
h1=hist(xin[y==0], col=rgb(0,1,0,1/4), xlim=c(0,4),ylim=c(0,30))
h2=hist(xin[y==1], col=rgb(0,0,1,1/4), xlim=c(0,4),ylim=c(0,30), add=T)
abline(v=cut)
error=.5*pnorm(cut,mu2,s2)+.5*(1-pnorm(cut,mu1,s1))
error
```

```
## [1] 0.197773
```

Step 2 Cost Functions and Model Output Function

Typeset the equations for the negative log likelihood (NLL) and squared error (SE) cost functions. After documenting the calculations, implement (complete stubs) the code. The cost functions take the entire x and y datasets as input (plus other arguments) and should be vectorized. Make sure to divide your total cost by the number of samples.

Typeset the equation for the model output probabilities. The model output function (f.prop) takes the full x dataset, along with model parameters and computes an output probability for every sample. Your code should be vectorized.

Logistic output

$$h(X; b, \mathbf{w}) = \sigma(b + X\mathbf{w})$$

The cost functions are: **Squared Error Cost**

$$\begin{aligned} C(b, \mathbf{w}) &= \frac{1}{2m}\|h(\mathbf{x}) - T\|^2 \\ &= \frac{1}{2m}\sum_{i=1}^{m}(h(\mathbf{x}^{(i)}) - t^{(i)})^2 \end{aligned}$$

Logistic Cost NLL

$$C(b, \mathbf{w}) = -\frac{1}{m}\sum_{i=1}^{m}\left(t^{(i)} \ln h(\mathbf{x}^{(i)}) + (1-t)^{(i)} \ln(1 - h(\mathbf{x}^{(i)}))\right)$$

Step 3 gradient calculations with Numerical Check

Derive and typeset the gradient equations (w.r.t. model parameters) for both cost functions. After documenting the calculations, implement (complete stubs) the code to return the gradients in a function called b.prop. b.prop takes as input the entire x and y datasets, yhat (predictions returned from f.prop) and a boolean called neg.log, when this is TRUE b.prop should return gradient for NLL otherwise SE. Implement your gradient code in a vectorized solution. Make sure to divide your total gradient by the number of samples.

To test your gradient functions, implement a function called num.gradient which takes as input the cost function,x,y,b,w,a tolerance (eps) and a boolean to indicate which cost function to use.

Compare the numerical gradient of your cost functions with the output of your gradient functions.

Also implement a function called predict that will take as input b,w, and X and return predictions using the logistic model that you can use to predict outcomes using your model and classify points with predicted probability >0.5 as 1 else 0.

SE Gradient

$$\begin{aligned} \frac{\partial C}{\partial \theta} &= \frac{1}{m}\sum_{i=1}^{m}(h(\mathbf{x}^{(i)}) - t^{(i)})\frac{\partial h^{(i)}}{\partial \theta} \\ \frac{\partial h}{\partial b} &= \frac{1}{m}\sum_{i=1}^{m} e_i \sigma_i (1 - \sigma_i) \\ \frac{\partial h}{\partial \mathbf{w}} &= \frac{1}{m}\sum_{i=1}^{m} e_i \sigma_i (1 - \sigma_i)\mathbf{x}^{(i)} \end{aligned}$$

Letting $e^{(i)} = h(\mathbf{x}^{(i)}) - t^{(i)}$ using $e^{(i)} \in \mathbb{R}$ gives

NLL Gradient

$$\begin{aligned} \frac{\partial C}{\partial b} &= \frac{1}{m}\sum_{i=1}^{m} e^{(i)} \\ \frac{\partial C}{\partial \mathbf{w}} &= \frac{1}{m}\sum_{i=1}^{m} e^{(i)}\mathbf{x}^{(i)} \end{aligned}$$

```
c(xin,y) %<-% generate.1d.dataset(50,1,.5,50,2,.7)
x=t(xin)
neg.logll=TRUE
```

```
yhat=f.prop(x,-0.02787209 ,0.16965187)
c(cost_val,db,dw) %<-% b.prop(x,y,yhat,TRUE)

num.gradient(cost,x,y,-0.02787209 ,0.16965187,1e-8,TRUE)
```

```
## $db
## [1] 0.0598427
##
## $dw
## [1] -0.1541206
```

```
c(xin,y) %<-% generate.1d.dataset(50,1,.5,50,2,.7)
x=t(xin)

neg.logll=TRUE
print("1d NLL compare of bprop to num grad case")
```

```
## [1] "1d NLL compare of bprop to num grad case"
```

```
c(b,w)%<-%c(runif(1,min=-1,max=1),runif(1,min=-1,max=1))
yhat=f.prop(x,b,w)
c(cost_val,db,dw) %<-% b.prop(x,y,yhat,TRUE)
ng=num.gradient(cost,x,y,b,w,1e-8,TRUE)
(ng$db-db)/db
```

```
## [1] 2.61594e-07
```

```
(ng$dw-dw)/dw
```

```
##              [,1]
## [1,] 1.265764e-08
```

```
print("1d SE compare of bprop to num grad case")
```

```
## [1] "1d SE compare of bprop to num grad case"
```

```
c(b,w)%<-%c(runif(1,min=-1,max=1),runif(1,min=-1,max=1))
yhat=f.prop(x,b,w)
c(cost_val,db,dw) %<-% b.prop(x,y,yhat,FALSE)
ng=num.gradient(cost,x,y,b,w,1e-8,FALSE)
(ng$db-db)/db
```

```
## [1] 1.181239e-08
```

```
(ng$dw-dw)/dw
```

```
##               [,1]
## [1,] -9.146167e-08
```

```
c(xin,y) %<-% generate.2d.dataset(n1=30,m1=c(1,1),c1=diag(c(1,1)),n2=70,m2=c(3,3),c2=c(1,1))
x=t(xin)
c(b,w)%<-%list(runif(1,min=-1,max=1),runif(2,min=-1,max=1))

yhat=f.prop(x,b,w)
c(cost_val,db,dw) %<-% b.prop(x,y,yhat,TRUE)
ng=num.gradient(cost,x,y,b,w,1e-8,TRUE)
print("2d NLL compare of bprop to num grad case")
```

```
## [1] "2d NLL compare of bprop to num grad case"
```

```
(ng$db-db)/db
```

```
## [1] -4.223489e-08
```

```
(ng$dw-dw)/dw
```

```
##               [,1]         [,2]
## [1,] 1.250186e-08 9.867699e-10
```

```
# 2d SE
print("2d SE compare of bprop to num grad case")
```

```
## [1] "2d SE compare of bprop to num grad case"
```

```
yhat=f.prop(x,b,w)
c(cost_val,db,dw) %<-% b.prop(x,y,yhat,FALSE)
ng=num.gradient(cost,x,y,b,w,1e-8,FALSE)
print("2d case")
```

```
## [1] "2d case"
```

```
(ng$db-db)/db
```

```
## [1] -1.18258e-08
```

```
(ng$dw-dw)/dw
```

```
##               [,1]          [,2]
## [1,] 3.126291e-09 -4.998107e-09
```

Step 4 Optimizer/Gradient Descent

Next code the log.fit stub. This function should include a learning rate parameter, a flag indicating which cost function to use and the number of iterations. This function should return a complete list of generated data: - history of (b,w) - history of cost - history of gradient norm - Use print fun to display code in log.fit

```
print.fun('log.fit')
```

```
## [1] "log.fit (x, y, neg.logll, eta, max.its, tol, b.init = 0, w.init = 0) "
## {
##     trace = list()
##     b = b.init
##     w = matrix(w.init, nrow = 1, ncol = nrow(x))
##     for (i in 1:max.its) {
##         yhat = f.prop(x, b, w)
##         c(cost, db, dw) %<-% b.prop(x, y, yhat, neg.logll)
##         if (is.nan(cost))
##             print(paste(b, w))
##         trace[[i]] = list(cost = cost, b = b, w = w, db = db,
##             dw = dw)
##         w = w - eta * dw
##         b = b - eta * db
##     }
##     trace
## }
```

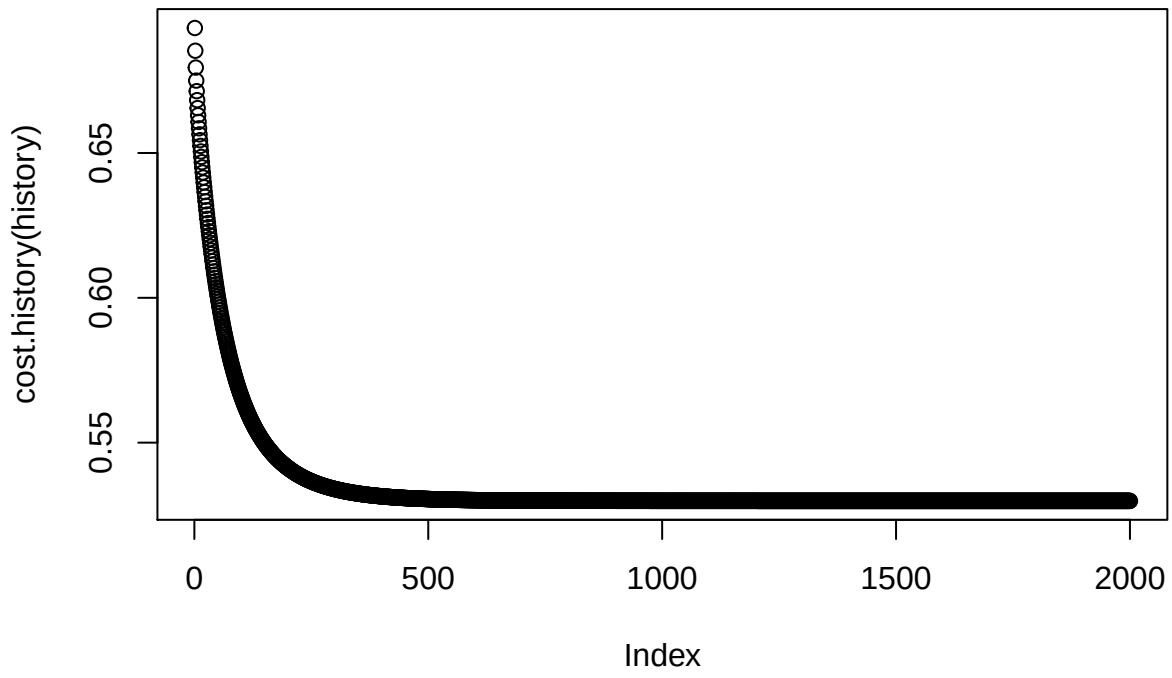
Step 5 Using the data set from step 1 build a model using your log.fit function and optimize the NLL cost function. use 2000 iterations and a learning rate eta of 0.25

- Compute the accuracy of the final model on the training plot, how does it compare to the bayes error from step 1?
- plot the cost history of the solution
- plot weight history
- plot gradient history and performance history of the model
- plot a contour plot of the b values, w values and cost function using the contour plot in r with 50 levels and draw a line showing the weight history on top of the contour plot in red using lines, type='o' and pch=16, col='red'

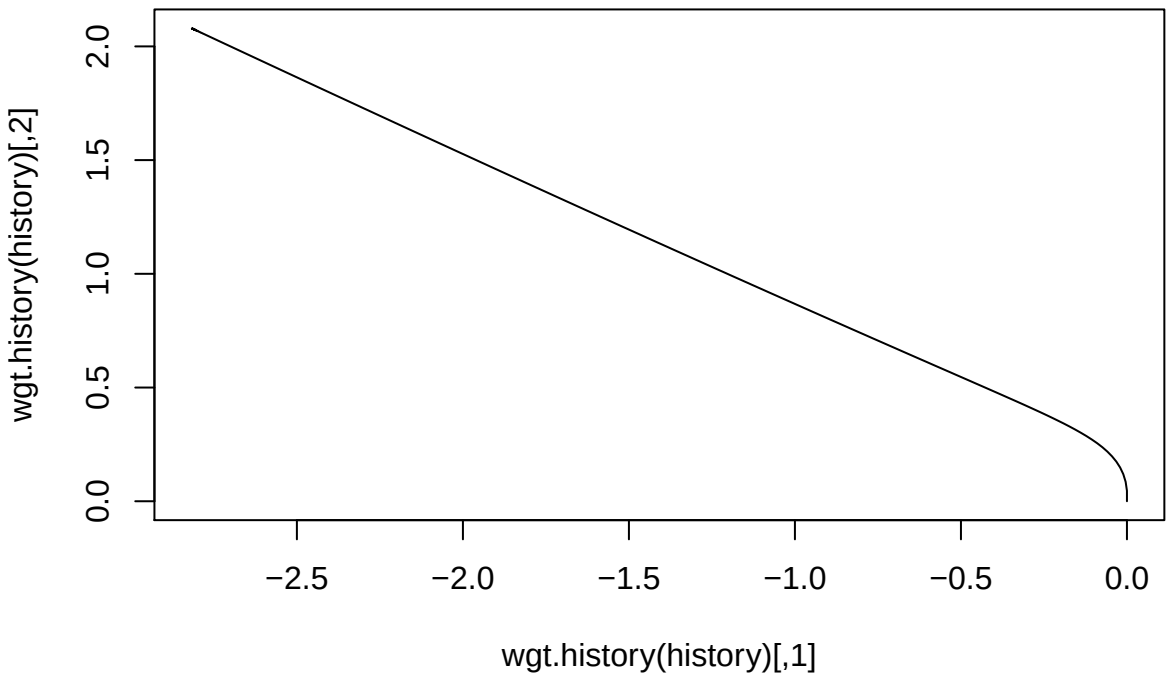
```
c(xin,y) %<-% generate.1d.dataset(50,1,.5,50,2,.7)

x=t(xin)
lr=.25
neg.logll=TRUE

history = log.fit(x,y,neg.logll=neg.logll,eta=lr,max.its=2000,tol=.00001)
c(final.cost,b,w,db,dw) %<-% history[[length(history)]]
print(accuracy(xin,y,b,w))
```

```
## [1] 0.75
```

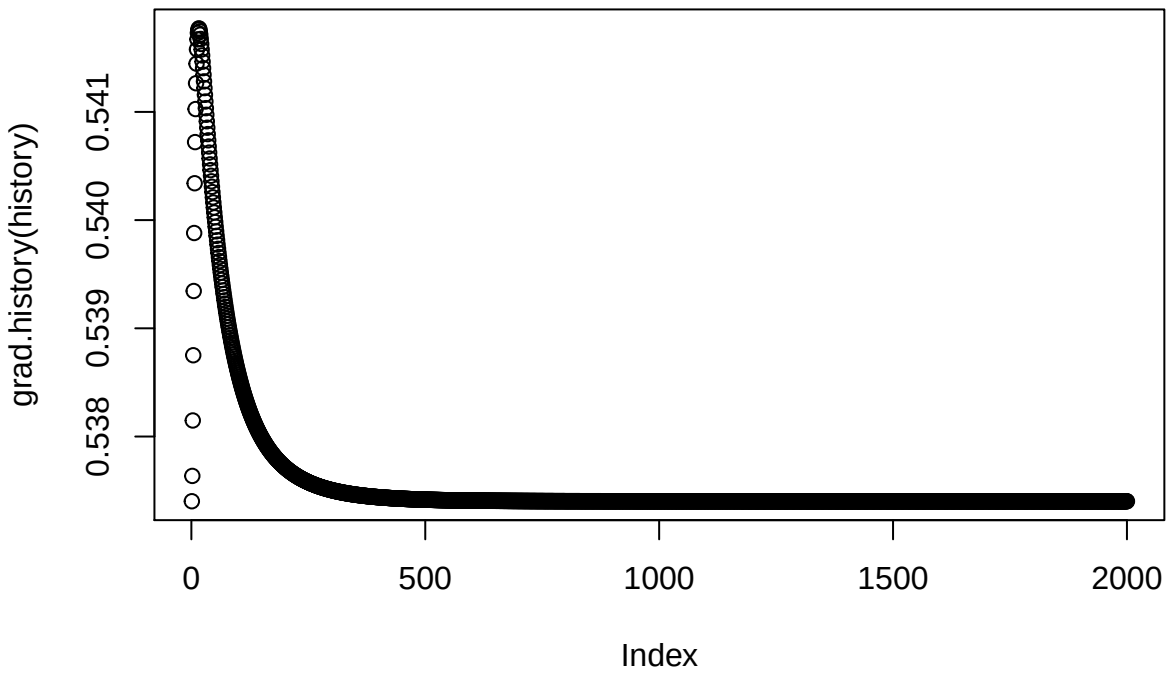
```
plot(cost.history(history))
```



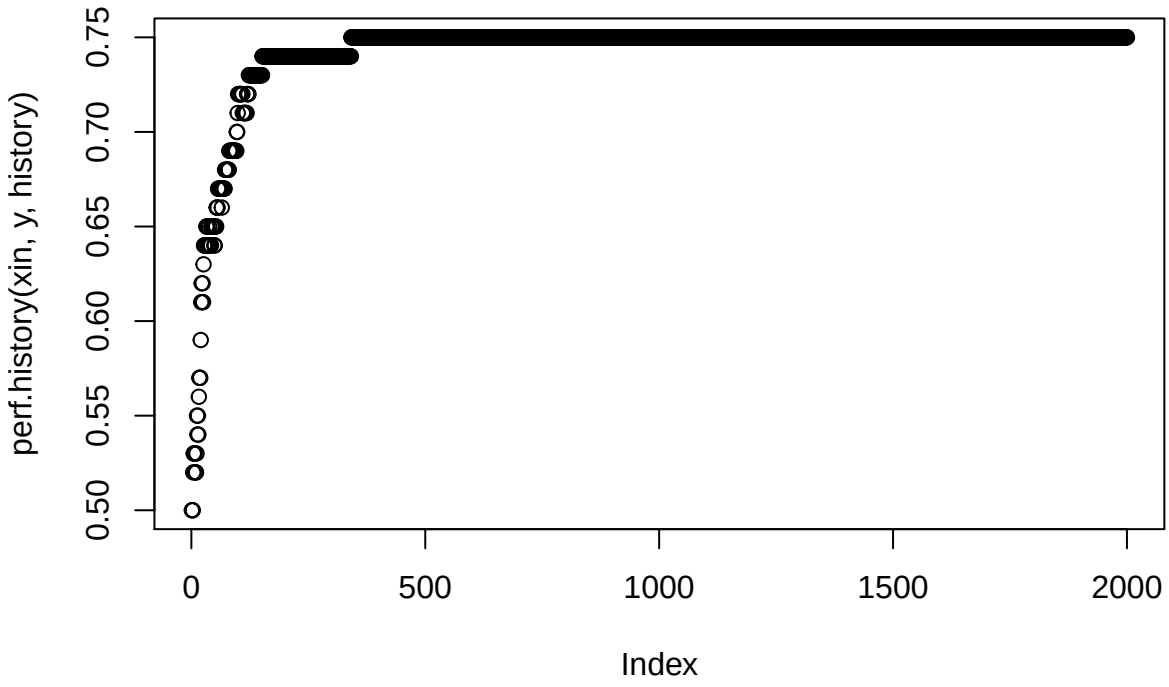

```
plot(wgt.history(history),type="l",pch=16)
```



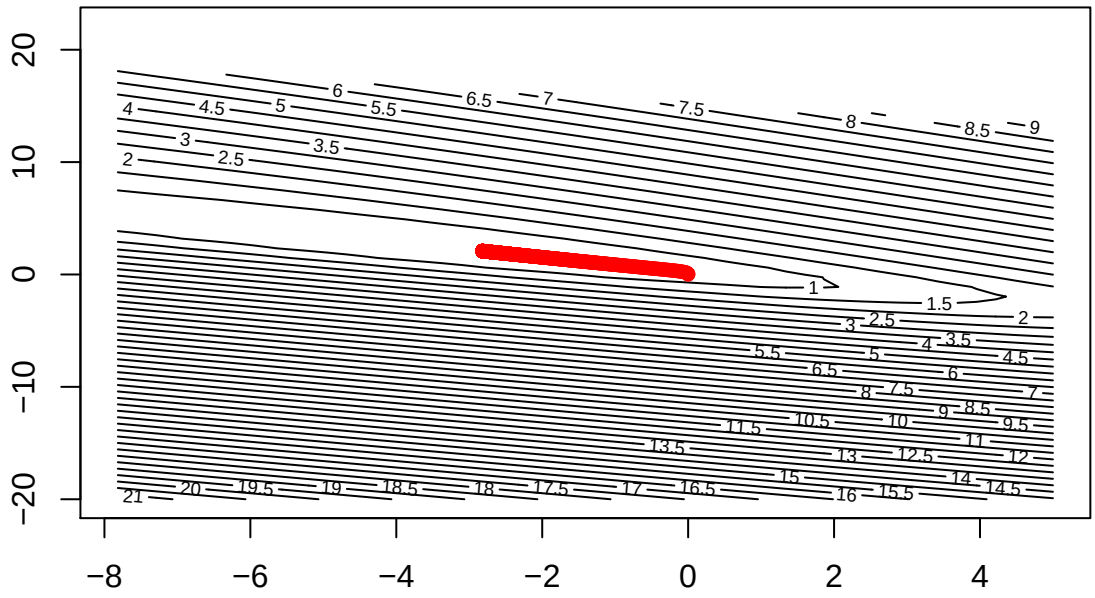

```
plot(grad.history(history))
```




```
plot(perf.history(xin,y,history))
```




```
vals=eval.grid(history,cost.func,b.margin = 5, w.margin = 20)
contour(vals$bv,vals$wv,vals$value,nlevels=50)
lines(wgt.history(history),type="o",pch=16,col="red")
```



Step 6

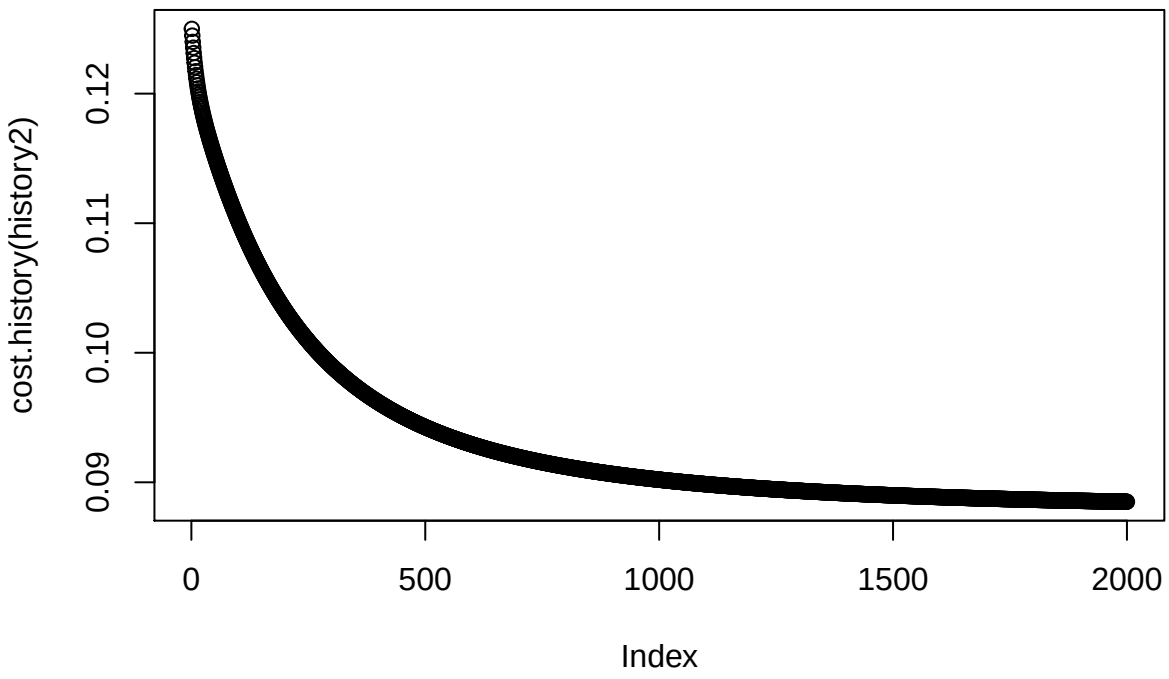
Using the data set from step 1 build a model using your log.fit function and optimize the SE (squared error) cost function. use 2000 iterations and a learning rate eta of 0.25.

- Compute the accuracy of the final model on the training plot
- plot the cost history of the solution
- plot weight history
- plot gradient history and performance history of the model
- plot a contour plot of the b values, w values and cost function using the contour plot in r with 50 levels
- plot a contour plot of the b values, w values and cost function using the contour plot in r with 50 levels and draw a line showing the weight history on top of the contour plot in red using lines, type='o' and pch=16, col='red'

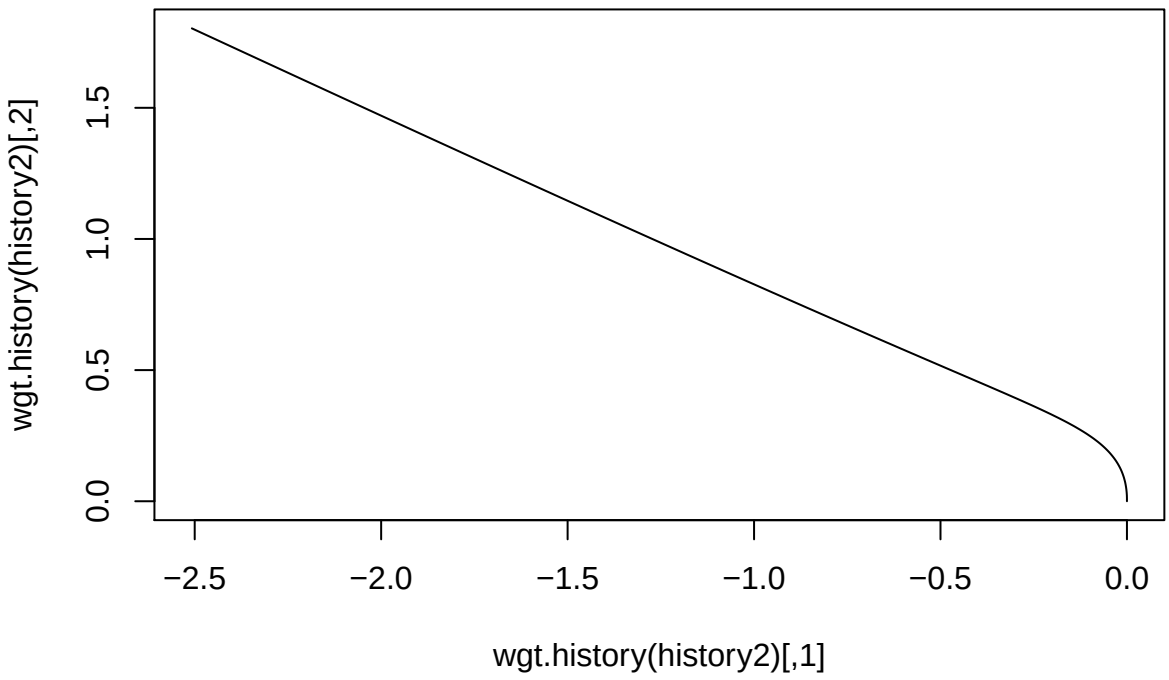
```
x=t(xin)
lr=.25
neg.logll=FALSE

history2 = log.fit(x,y,neg.logll=neg.logll,eta=lr,max.its=2000,tol=.00001)
c(final.cost,b,w,db,dw) %<-% history2[[length(history2)]]
print(accuracy(xin,y,b,w))
```

```
## [1] 0.76
```

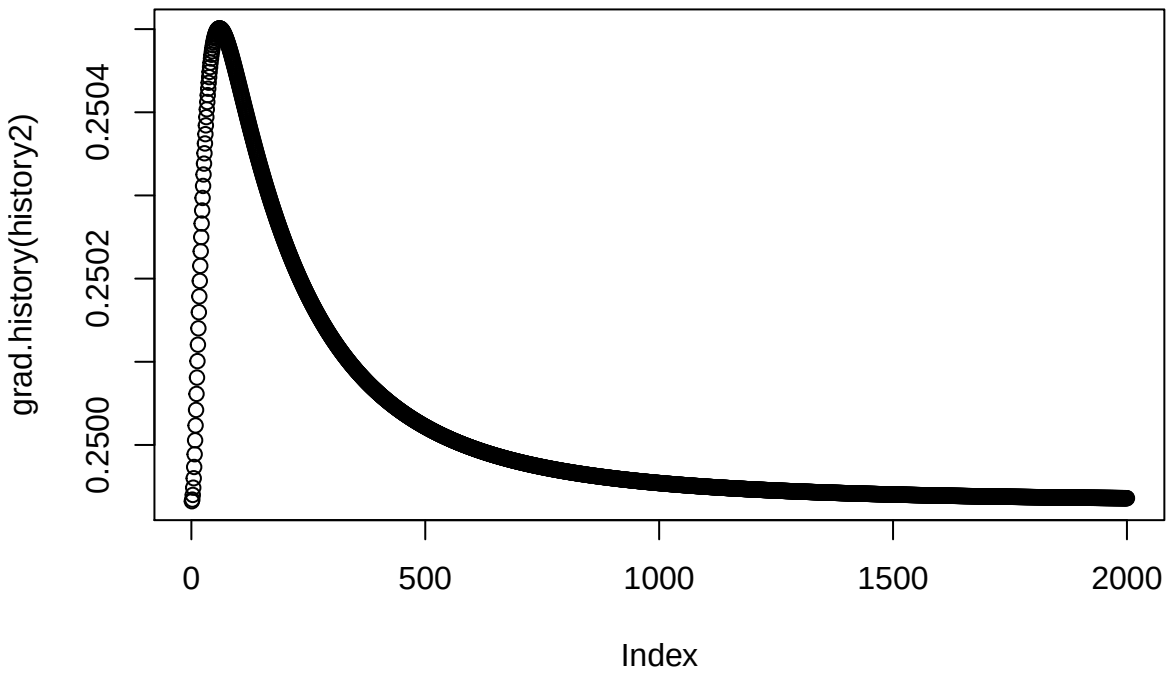
```
plot(cost.history(history2))
```



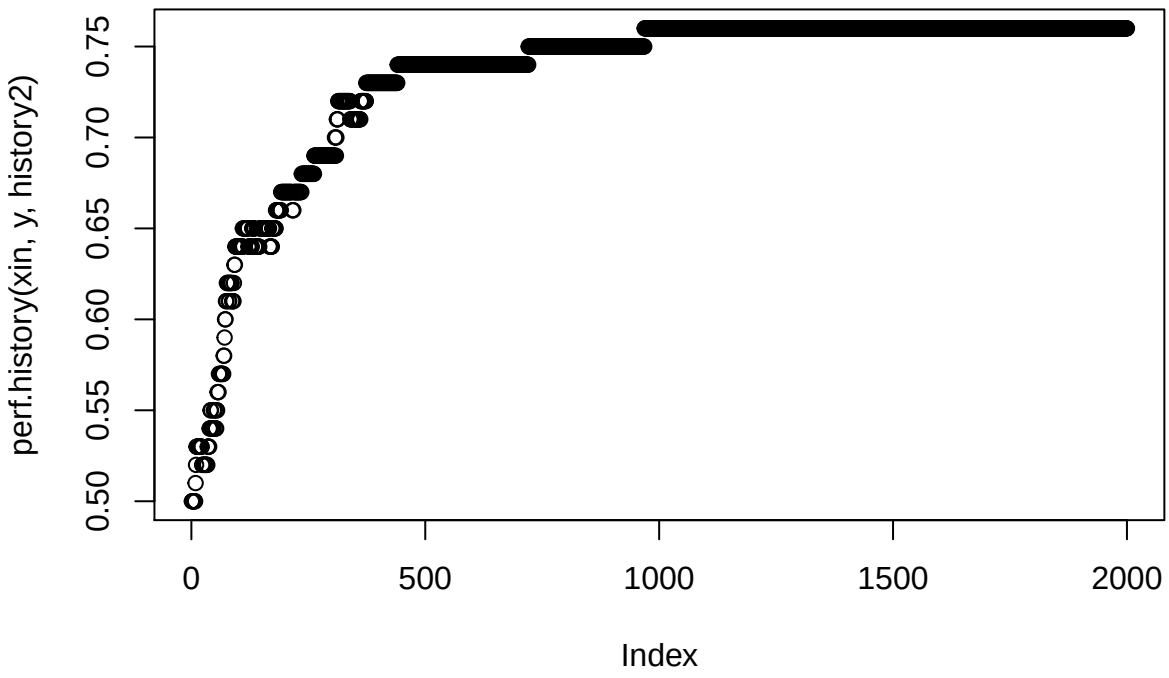

```
plot(wgt.history(history2),type="l",pch=16)
```



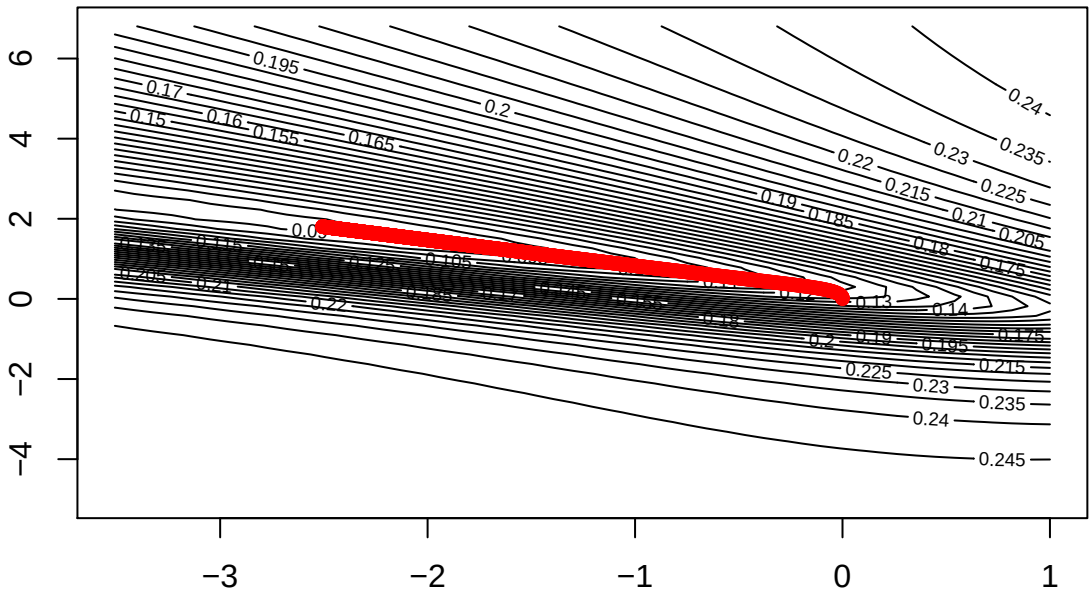

```
plot(grad.history(history2))
```



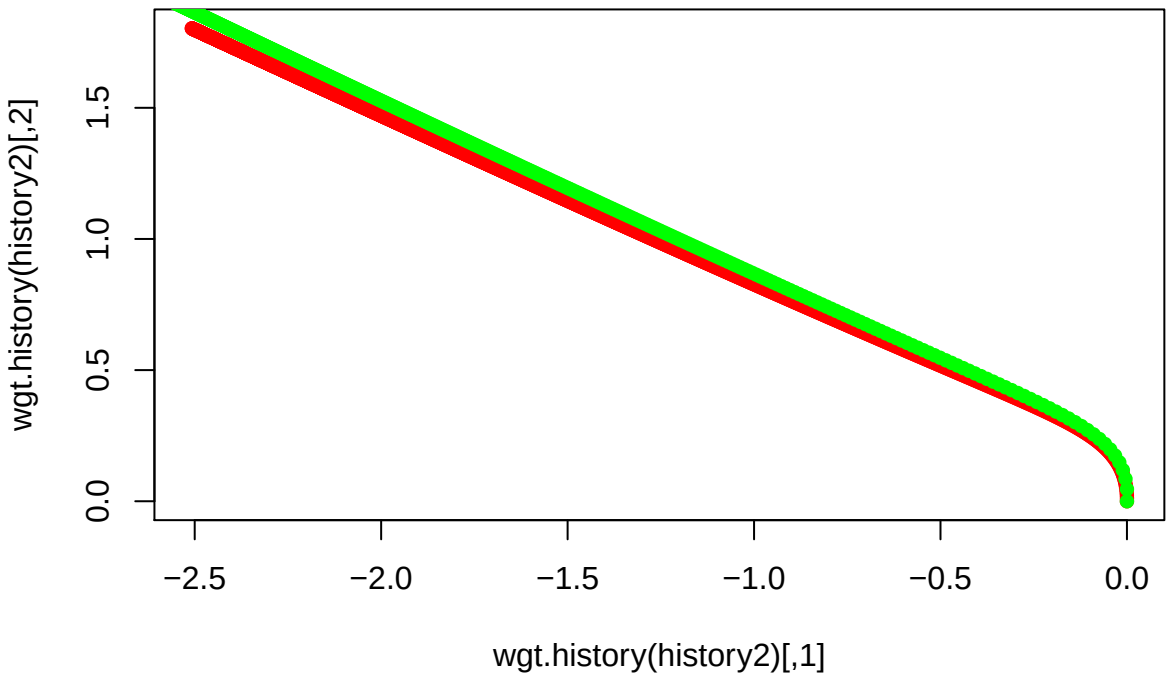

```
plot(perf.history(xin,y,history2))
```




```
vals=eval.grid(history2,cost.func,b.margin = 1, w.margin = 5)
contour(vals$bv,vals$wv,vals$value,nlevels=50)
lines(wgt.history(history2),type="o",pch=16,col="red")
```



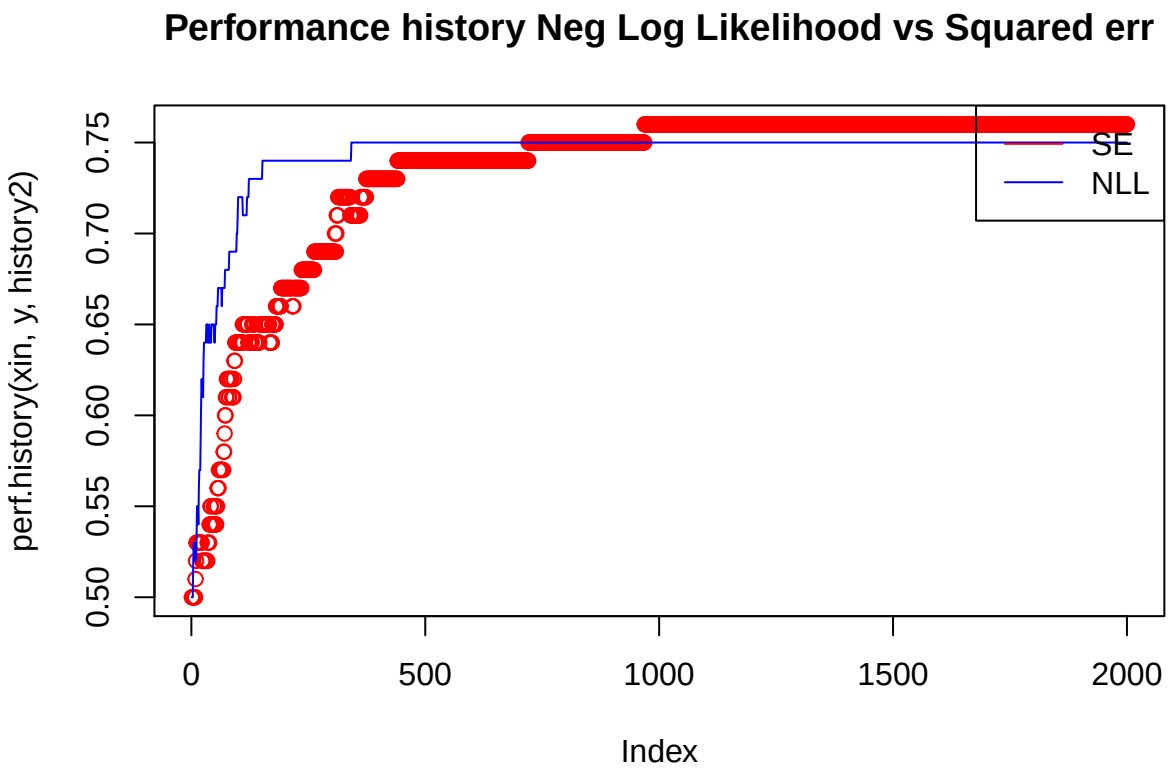

```
plot(wgt.history(history2),type="o",pch=16,col="red")
points(wgt.history(history),type="o",pch=16,col="green")
```



Step 7 Compare performance of NLL and SE

Plot the performance histories of the 2 cost functions side by side with different colors and legend (red for SE and blue line for NLL)

```
plot(perf.history(xin,y,history2),col='red',main='Performance history Neg Log Likelihood vs Squared err
lines(perf.history(xin,y,history),col='blue')
legend("topright", legend = c("SE","NLL"),
  col = c('red','blue'),lty=1,lwd=1 )
```



Step 8

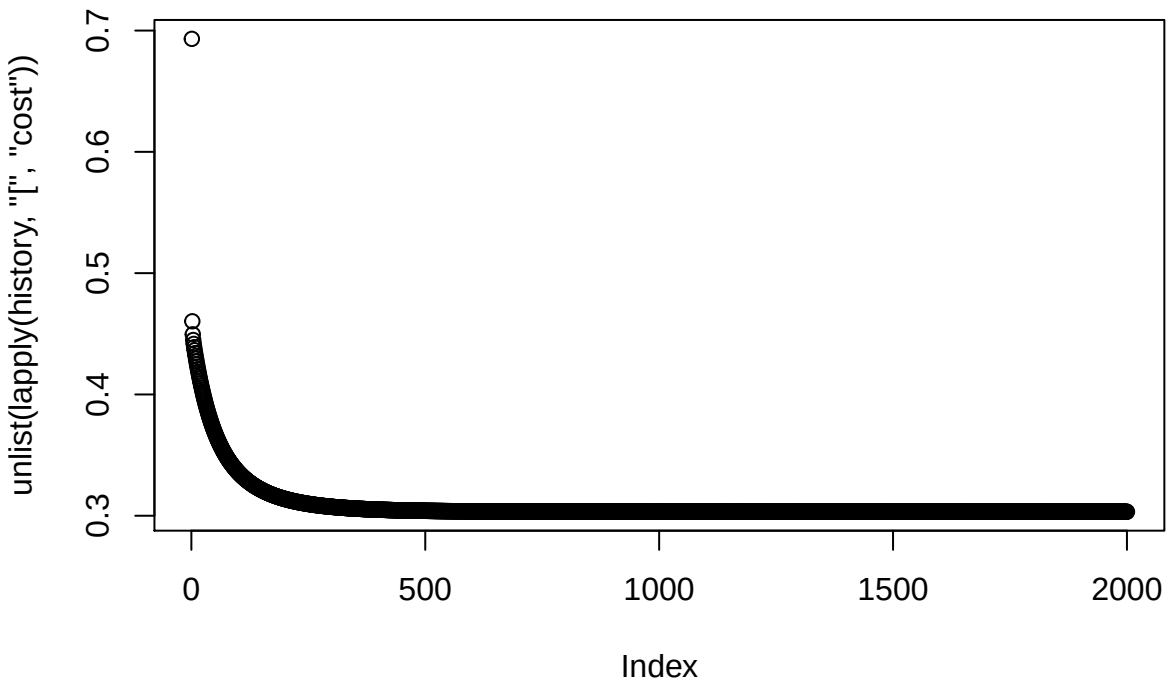
Generate a 2-d data set using generate.2d.dataset function and use log.fit (gradient descent) to build a model using the NLL cost function. As this data set is 2-d instead of creating a contour plot plot the decision boundary using a grid of values for the x1 and x2 in the data set as the axis. Use a learning rate, eta, of .25 and 2000 iterations.

- Compute the accuracy of the final model on the training plot
- plot the cost history of the solution
- plot weight history
- plot gradient history and performance history of the model
- plot the decision boundary for x1 and x2 variables in the data
- compute the accuracy of the model on 10 out of sample data sets generated using the same function and compute the mean accuracy out of sample. Is it lower than training accuracy?

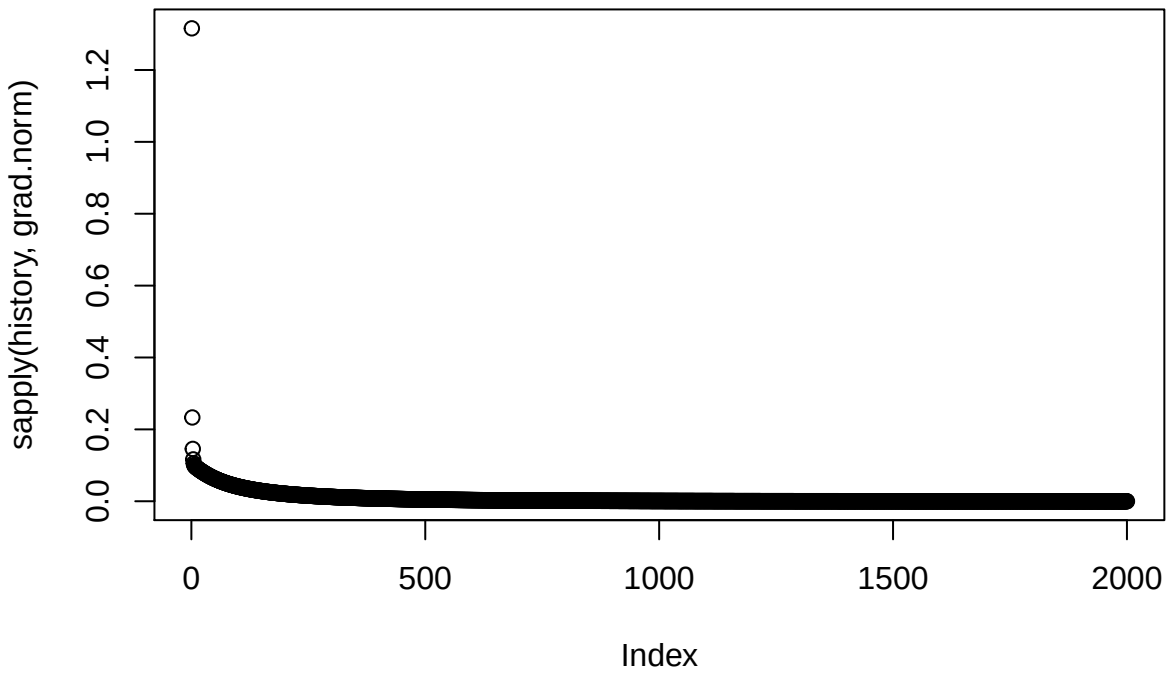
```
# 2d,
  c(xin,y) %<-% generate.2d.dataset(n1=30,m1=c(1,1),c1=diag(c(1,1)),n2=70,m2=c(3,3),c2=c(1,1))
  x=t(xin)
  history = log.fit(x,y,neg.logll=TRUE,eta=.25,max.its=2000)
  c(final.cost,b,w,db,dw) %<-% history[[length(history)]]
  print(accuracy(xin,y,b,w))
```

```
## [1] 0.86
```

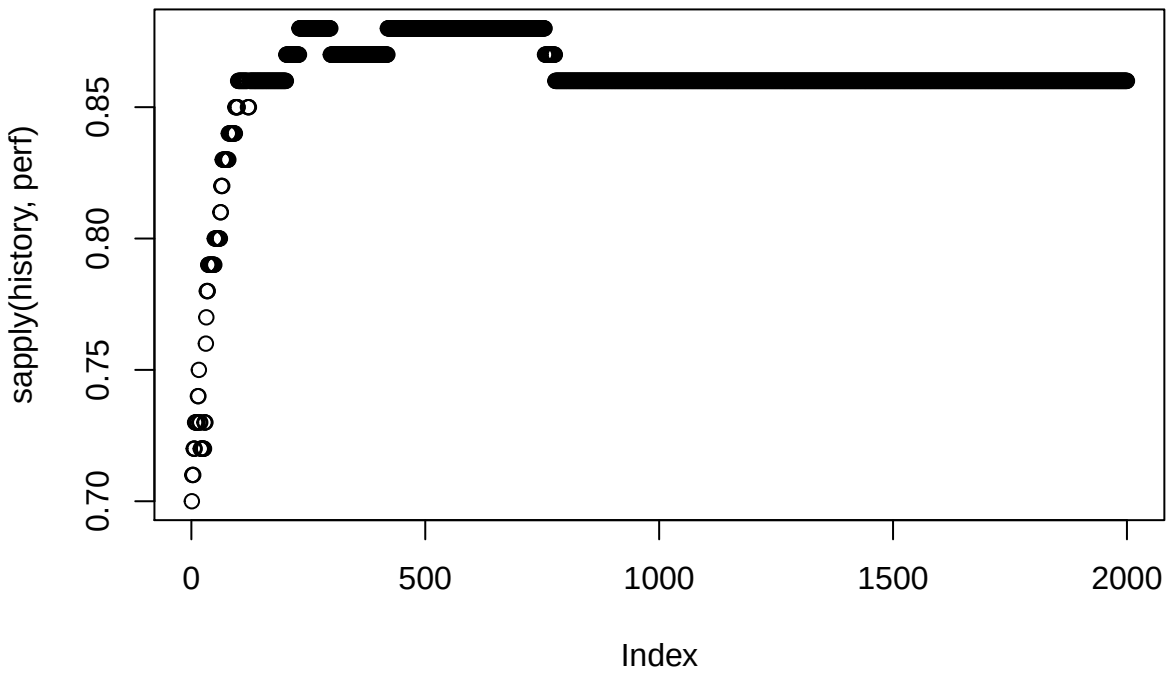
```
plot.cost.history(history)
```



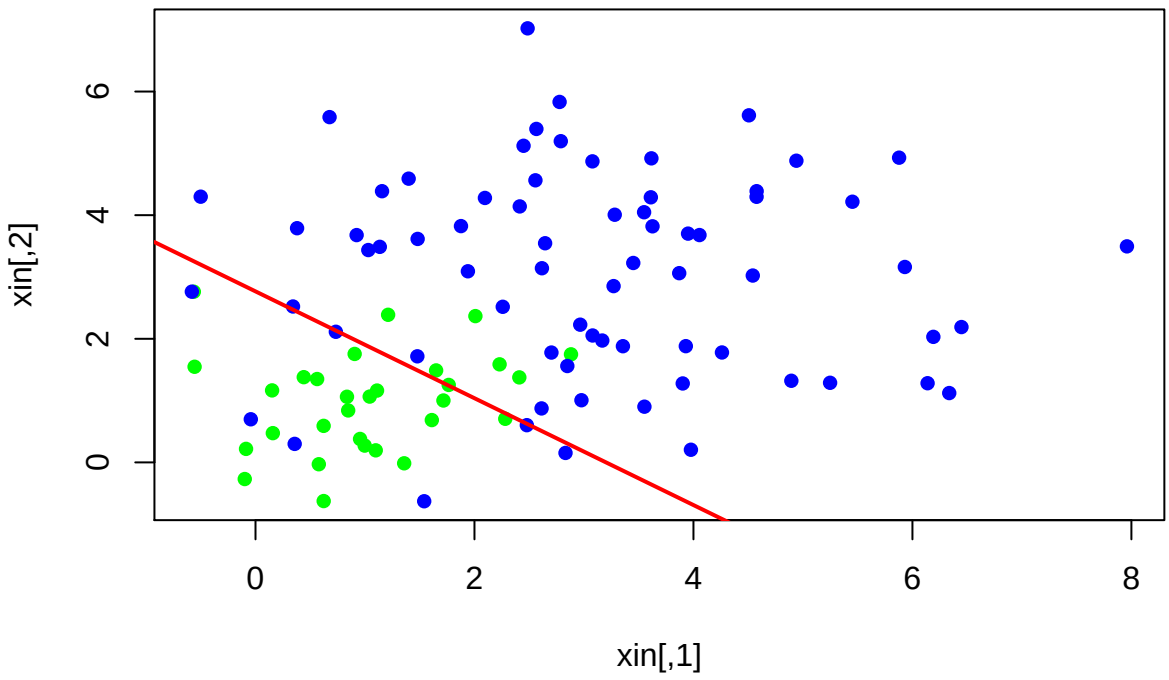

```
plot.grad.history(history)
```




```
plot.perf.history(xin,y,history)
```




```
b = history[[length(history)]]$b
w = history[[length(history)]]$w
plot(xin,col=c("green","blue")[y+1],pch=16)
abline(logistic.2d.line(b,w),col="red",lwd=2)
```




```
  results=vector(mode='list')
  for (i in 1:10){
    c(test,ytest) %<-% generate.2d.dataset(n1=30,m1=c(1,1),c1=diag(c(1,1)),n2=70,m2=c(3,3),c2=c(1,1))
    results[[i]]=accuracy(test,ytest,b,w)
  }
  print(paste('Using 10 trials mean out of sample accuracy of model is:',mean(unlist(results))))
```

```
## [1] "Using 10 trials mean out of sample accuracy of model is: 0.841"
```

```
  print(paste('The training accuracy of model is:',accuracy(xin,y,b,w)))
```

```
## [1] "The training accuracy of model is: 0.86"
```

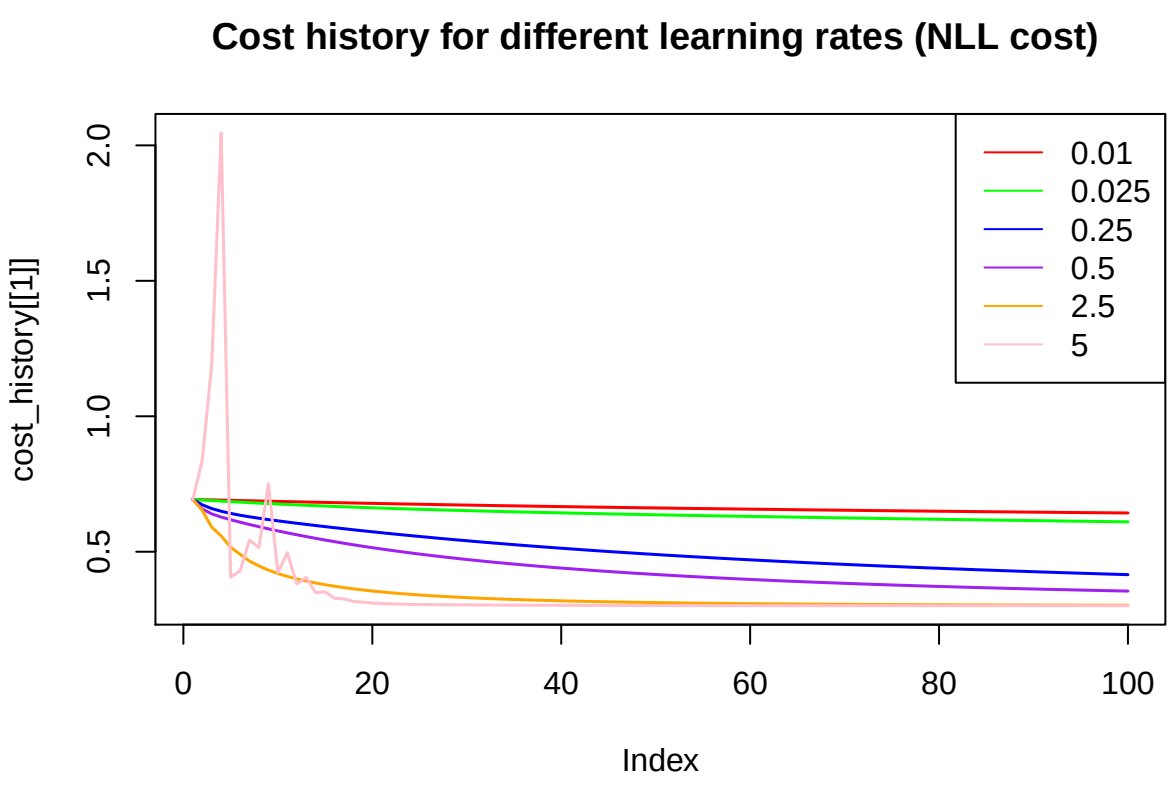
Step 9

Setting the learning rate is one of the most important parameter in models. In this step you will run an experiment to evaluate a range of learning rates using the NLL cost and plot the cost histories.

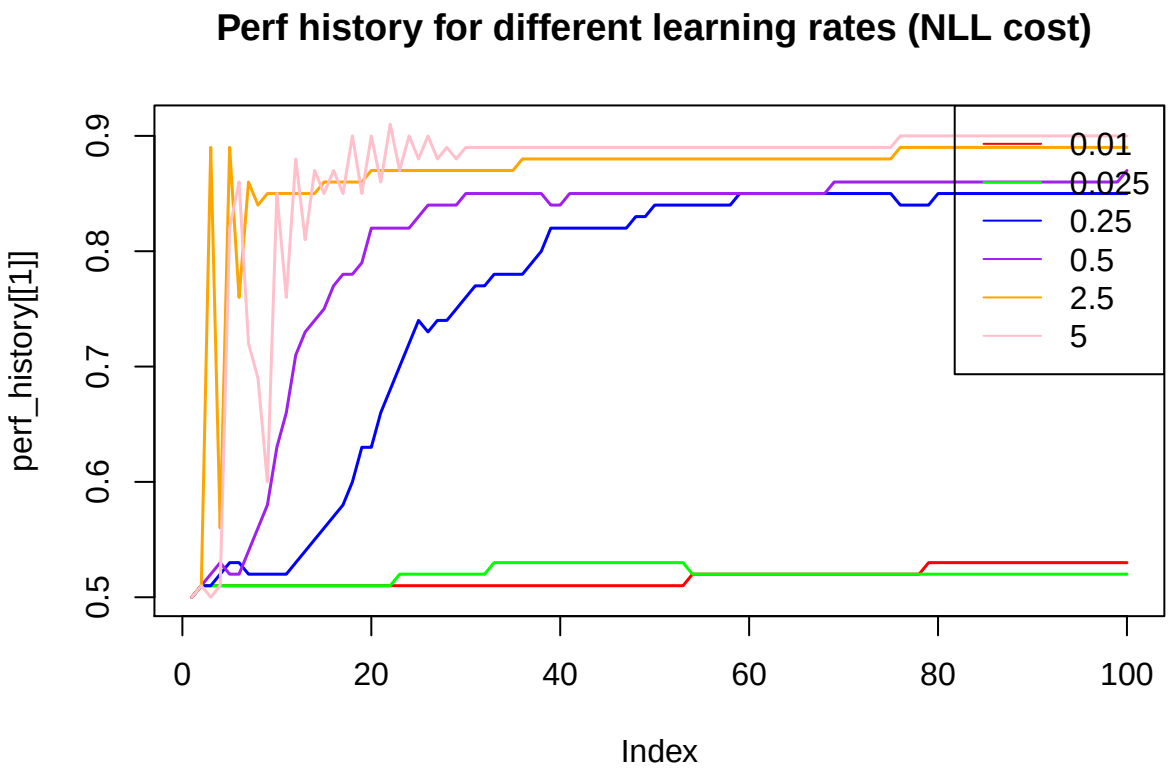
- Use the same 1-d dataset used earlier for training and run your gradient descent optimizing NLL and save histories for each learning rate. Use learning rates of .01,.025,2.5, and 5 for your experiments.
- Plot the cost histories for each learning rate
- Create a contour plot and overlay it with a red line for the wgt history for the learning rate of 5.

```
cost_history=vector(mode='list')
wgt_history=vector(mode='list')
perf_history=vector(mode='list')
hist=vector(mode='list')
c(xin,y) %<-% generate.1d.dataset(50,1,.5,50,2,.7)
neg.logll=TRUE
```

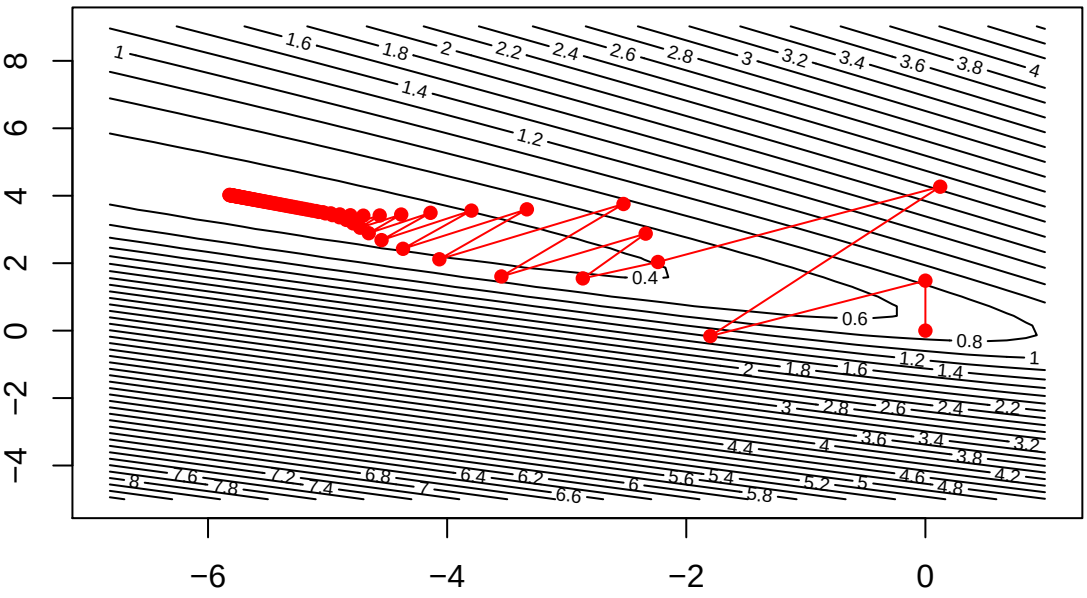
```
counter=1
lr=c(.01,.025,.25,.5,2.5,5)
for (i in lr){
  x=t(xin)
  history = log.fit(x,y,neg.logll=neg.logll,eta=i,max.its=100)
  c(final.cost,b,w,db,dw) %<-% history[[length(history)]]
  cost_history[[counter]]=cost.history(history)
  perf_history[[counter]]=perf.history(xin,y,history)

  wgt_history[[counter]]=wgt.history(history)
  hist[[counter]]=history
  counter=counter+1
}

colors <- c('red','green','blue','purple','orange','pink','black')
#linetype <- c(1:length(lr))
plot(cost_history[[1]],lwd=1.5,type='l',col=colors[1],main='Cost history for different learning rates (
for (i in 2:length(lr)) lines(cost_history[[i]],col=colors[i],lwd=1.5)
legend("topright", legend = lr,
  col = colors,lwd=1 )
```




```
plot(perf_history[[1]],lwd=1.5,type='l',col=colors[1],main='Perf history for different learning rates (
for (i in 2:length(lr)) lines(perf_history[[i]],col=colors[i],lwd=1.5)
legend("topright", legend = lr,
  col = colors,lwd=1 )
```




```
vals=eval.grid(hist[[length(lr)]],cost.func,b.margin = 1, w.margin = 5)
contour(vals$bv,vals$wv,vals$value,nlevels=50)
lines(wgt.history(hist[[length(lr)]]),type="o",pch=16,col="red")
```

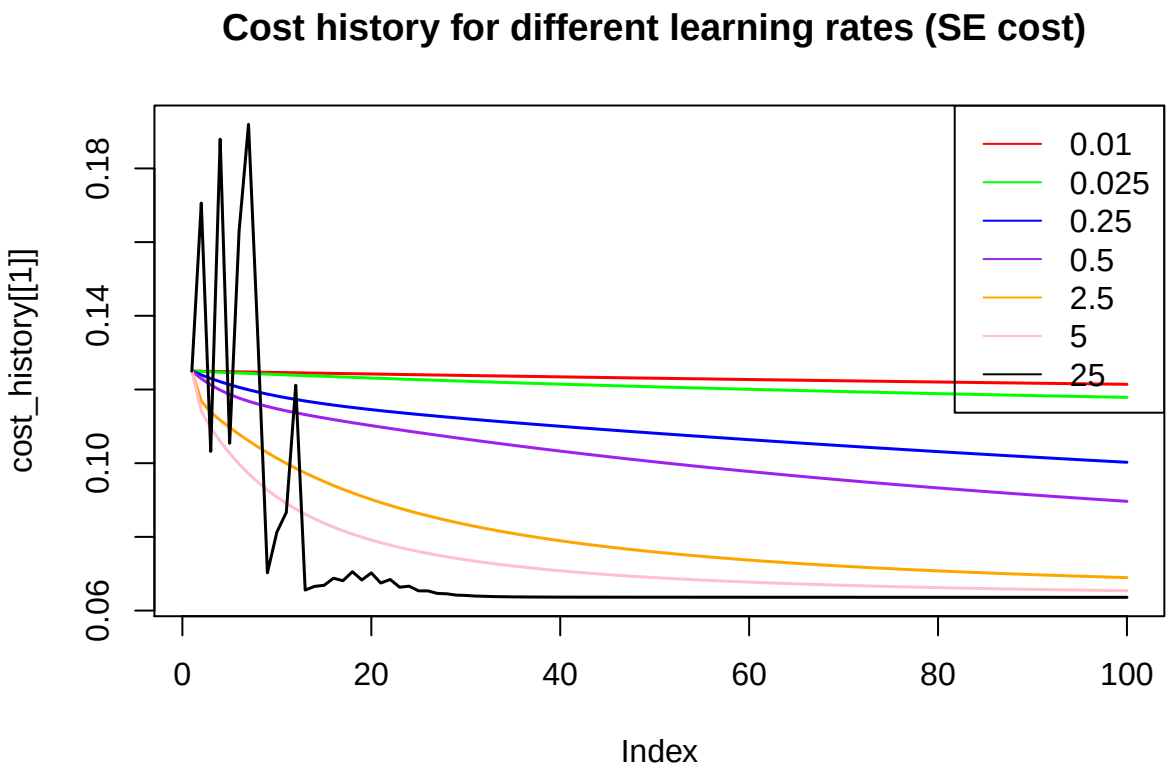


** Step 10 **

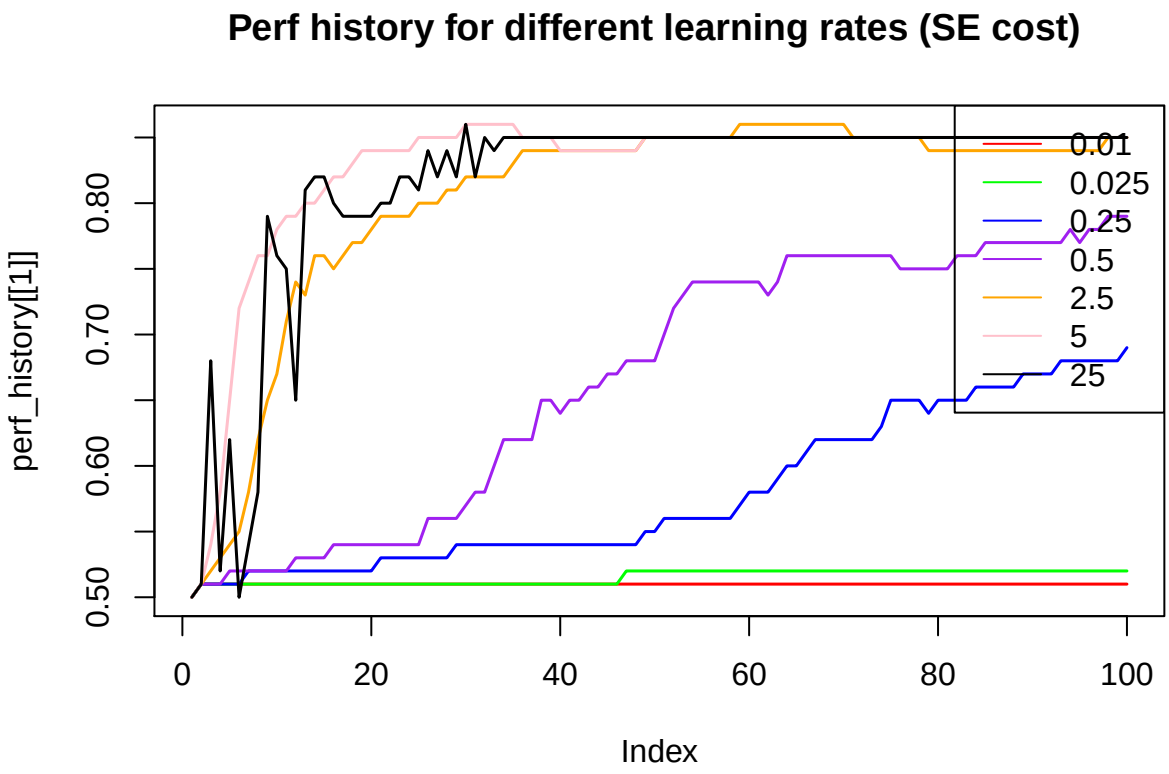
Repeat the learning rate exploration using SE cost function using the same experimental setup from step 9 except use a learning rate of 25 instead of 5. Produce the 2 plots for cost histories by learning rate and contour plot with wgt history red line for the learning rate of 25.

```
cost_history=vector(mode='list')
wgt_history=vector(mode='list')
perf_history=vector(mode='list')
hist=vector(mode='list')
c(xin,y) %<-% generate.1d.dataset(50,1,.5,50,2,.7)
neg.logll=FALSE
counter=1
lr=c(.01,.025,.25,.5,2.5,5,25)
for (i in lr){
  x=t(xin)
  history = log.fit(x,y,neg.logll=neg.logll,eta=i,max.its=100)
  c(final.cost,b,w,db,dw) %<-% history[[length(history)]]
  cost_history[[counter]]=cost.history(history)
  wgt_history[[counter]]=wgt.history(history)
  perf_history[[counter]]=perf.history(xin,y,history)
  hist[[counter]]=history
  counter=counter+1
}

colors <- c('red','green','blue','purple','orange','pink','black')
#linetype <- c(1:length(lr))
plot(cost_history[[1]],lwd=1.5,type='l',col=colors[1],main='Cost history for different learning rates (S
```

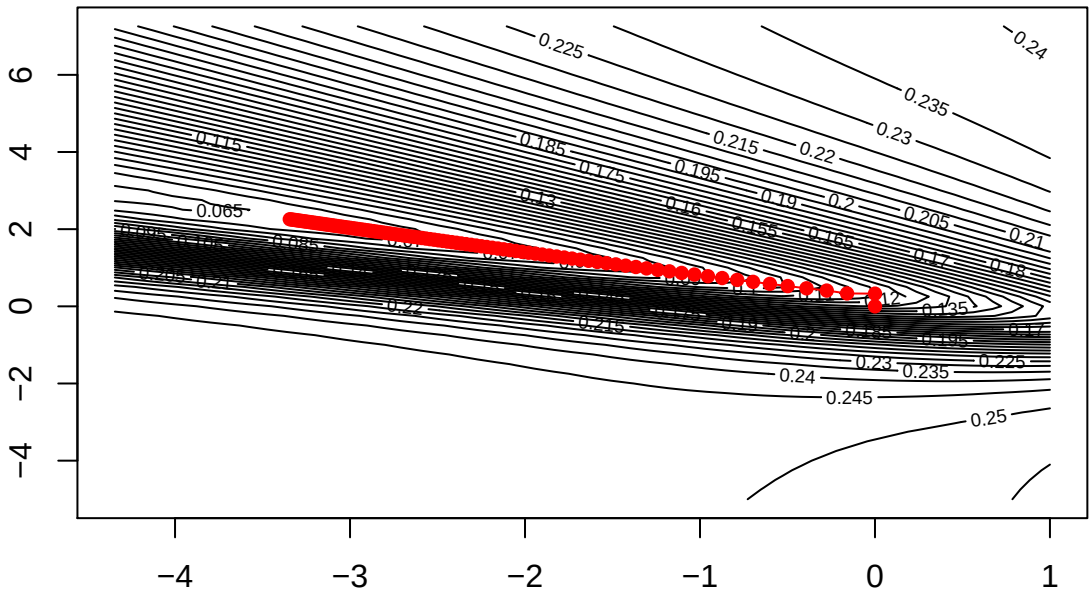
```
for (i in 2:length(lr)) lines(cost_history[[i]],col=colors[i],lwd=1.5)
legend("topright", legend = lr,
  col = colors,lwd=1 )
```




```
plot(perf_history[[1]],lwd=1.5,type='l',col=colors[1],main='Perf history for different learning rates (S
for (i in 2:length(lr)) lines(perf_history[[i]],col=colors[i],lwd=1.5)
legend("topright", legend = lr,
  col = colors,lwd=1 )
```




```
vals=eval.grid(hist[[6]],cost.func,b.margin = 1, w.margin = 5)
contour(vals$bv,vals$wv,vals$value,nlevels=50)
lines(wgt.history(hist[[6]]),type="o",pch=16,col="red")
```


0.24
0.235
0.23
0.225
0.215
0.22
0.195
0.19
0.2
0.205
0.21
0.185
0.175
0.18
0.115
0.065
0.135
0.24
0.235
0.23
0.225
0.245
0.25
6
4
2
0
−2
−4
−4
−3
−2
−1
0
1