week 7 notes

Brady Miller

Table of contents

Tuesday, Feb 21

❗ TIL

Include a *very brief* summary of what you learnt in this class here.
Today, I learnt the following concepts in class:

1. General regularization/shrinkage estimators
2. LASSO regression estimator
3. Gradient descent

```r
# importing necessary libraries and the data set utilized in class
library(ISLR2)
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(tidyr)
library(purrr)
library(readr)
library(glmnet)
```

```
Loading required package: Matrix

Attaching package: 'Matrix'

The following objects are masked from 'package:tidyr':

    expand, pack, unpack

Loaded glmnet 4.1-6
```

```r
library(caret)
```

```
Loading required package: ggplot2

Loading required package: lattice

Attaching package: 'caret'

The following object is masked from 'package:purrr':

    lift
```

```r
library(car)
```

```
Loading required package: carData


Attaching package: 'car'


The following object is masked from 'package:purrr':

    some


The following object is masked from 'package:dplyr':

    recode
```

```r
library(torch)

df <- Boston
attach(Boston)
```

February 21st

Regularization/Shrinkage estimators

Objective function defined below:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p + \epsilon$$

The least-squares objective selects the model with the smallest residual standard error

$$L(\beta_0, \beta_2, \dots, \beta_p) = SS_{Res} = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{1,i} - \dots - \beta_p x_{p,i})^2$$

The solution to this problem is denoted as follows…

$$(b_1, b_2, \dots, b_p) = \underset{\beta_1 \dots \beta_p}{\arg\min}\, L(\beta_0, \beta_1, \dots, \beta_p)$$

* Don't always want every variable from a data set in our final model

- To select only a subset of these variables in our final model, we can include a penalty term (include penalty term that doesn't have the intercept)
- Below is the penalty term

$$p_\lambda(\beta_1, \dots, \beta_p)$$

* This penalty term favors solutions which select smaller subset of the variables (sparser solutions),as some variables may not be 'important' to the final model.

- When we include the penalty term, the objective function becomes…

$$L(\beta_0, \beta_1, \dots, \beta_p) = L(\beta_0, \beta_2, \dots, \beta_p) + p_\lambda(\beta_1, \dots, \beta_p)$$

In class we mentioned some of the most common penalty functions which are:

1. Ridge Regression estimator
$$p_\lambda = \beta_1^2 + \beta_2^2 + \dots + \beta_p^2$$
2. LASSO regression estimator
$$p_\lambda = |\beta_1| + |\beta_2| + \dots + |\beta_p|$$
3. General case in glmnet()
$$p_\lambda = |\beta_1|^\alpha + |\beta_2|^\alpha + \dots + |\beta_p|^\alpha$$

In the case of each penalty term, we can see that we want to find a solution which:

- Minimizes SS_{Res}, and
- Minimizes p_λ, which means that we want to find a solution which favors sparser solutions

How the penalty term impacts the objective function:

- After implementing the penalty function if any of the β_p turns out to be 0, it means that it doesn't have an impact on the model as you are multiplying the variable by 0 so it won't be included (for a change in that x_p, there is no change in the model) –> the variables associated with the zeroes are then dropped from the final model
- The variables that are co-linear are shrunk to 0, therefore eliminating those variables from the final model (deems that variable not important)

LASSO

Unlike lm(), the glmnet() function doesn't take in a formula

To use LASSO we can first rescale the variables so they are all on same scale

```r
full_model<- lm(medv ~., df)
X <- model.matrix(full_model)[,-1]
head(X)
```

```
     crim zn indus chas   nox    rm  age    dis rad tax ptratio lstat
1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3  4.98
2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8  9.14
3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8  4.03
4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7  2.94
5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7  5.33
6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7  5.21
```

```r
all_cols <- 1:ncol(X)
drop_scale <- c(4)
include_scale <- all_cols[-drop_scale]

for (i in include_scale) { X[,i] <- scale(X[,i]) }
head(X)
```

```
        crim         zn      indus chas        nox        rm        age
1 -0.4193669  0.2845483 -1.2866362    0 -0.1440749 0.4132629 -0.1198948
2 -0.4169267 -0.4872402 -0.5927944    0 -0.7395304 0.1940824  0.3668034
3 -0.4169290 -0.4872402 -0.5927944    0 -0.7395304 1.2814456 -0.2655490
4 -0.4163384 -0.4872402 -1.3055857    0 -0.8344581 1.0152978 -0.8090878
5 -0.4120741 -0.4872402 -1.3055857    0 -0.8344581 1.2273620 -0.5106743
6 -0.4166314 -0.4872402 -1.3055857    0 -0.8344581 0.2068916 -0.3508100
       dis        rad        tax    ptratio      lstat
1 0.140075 -0.9818712 -0.6659492 -1.4575580 -1.0744990
2 0.556609 -0.8670245 -0.9863534 -0.3027945 -0.4919525
3 0.556609 -0.8670245 -0.9863534 -0.3027945 -1.2075324
4 1.076671 -0.7521778 -1.1050216  0.1129203 -1.3601708
5 1.076671 -0.7521778 -1.1050216  0.1129203 -1.0254866
6 1.076671 -0.7521778 -1.1050216  0.1129203 -1.0422909
```

All values are now in same scale (between -3 and 3)

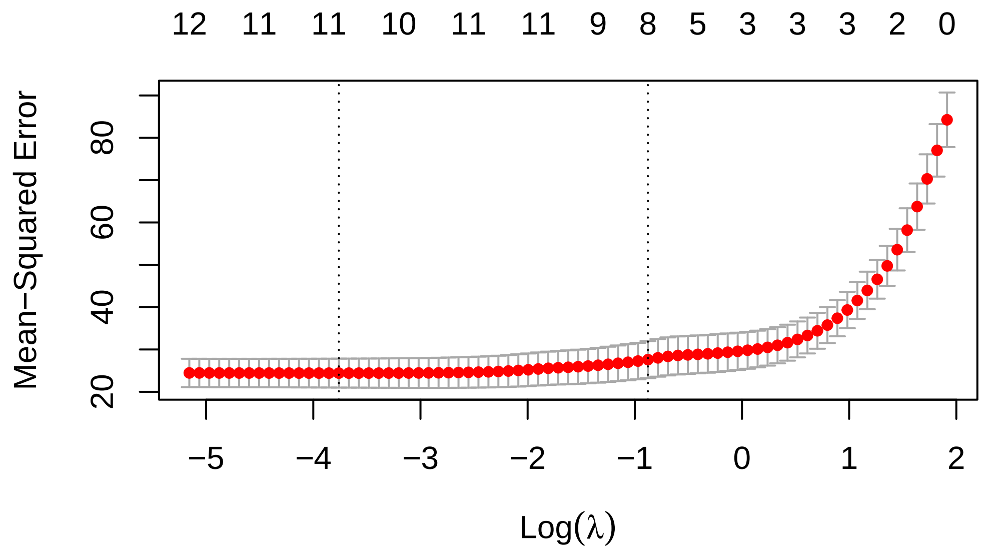
```r
y <- df$medv
```

```r
lasso <- cv.glmnet(X,y,alpha = 1)
# alpha is exponent for function
```

```r
lasso
```

```
Call:  cv.glmnet(x = X, y = y, alpha = 1)

Measure: Mean-Squared Error

    Lambda Index Measure    SE Nonzero
min 0.0233    62   24.42 3.416      10
1se 0.4159    31   27.60 4.378       8
```

```r
plot(lasso)
```



Plot explanation:

- For every lambda in range, computes the estimator
- plots mean squared error (sum of squared residual)
- The penalty we include depends on value of lambda –> different lambda value leads to different subset of variables selected

- As lambda increases, the effect that the penalty has on the solution is stronger (the value of p sub lambda also increases)
- If minimizing p_λ, want to drop more variables and sparser solutions
- As we go from right to left (lambda increases) the number of variables that are selected decreases (number of variables selected is along the top)
- Is a balancing act
- Near 0 penalty = select all variables & has lower mean squared error
- Introducing large penalty –> sparse solutions & has higher mean squared error

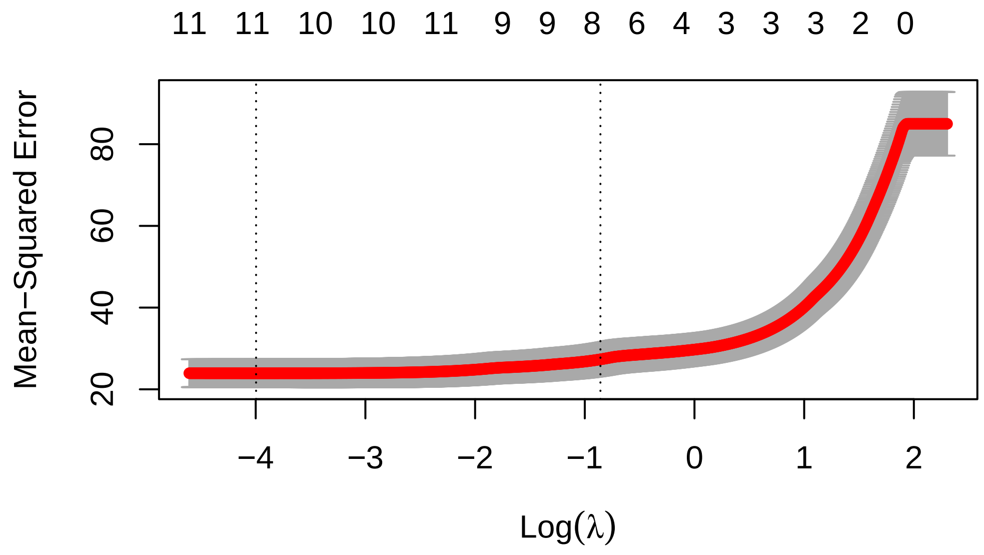
How to known what lambda value is appropriate…

- select the λ value right before where it spikes upwards (choose elbow point), as this is most stable solution

    1. R has algorithm presented in next code cell that chooses the elbow point that minimizes mean squared error

In the code below, we specifying sequence of values of lambda to search

```
lambdas <- 10 ^ seq(-2,1,length.out = 1000)
lasso <- cv.glmnet(X,y,alpha = 1,lambda = lambdas)
plot(lasso)
```



```
lasso_coef <- coef(lasso, s = "lambda.min")
# can do lambda.1se to choose different lambda that will result in different
# amount of variables chosen
selected_vars <- rownames(lasso_coef)[which(abs(lasso_coef) > 0)][-1]
# excludes the intercept term
lasso_coef
```

```
13 x 1 sparse Matrix of class "dgCMatrix"
                     s1
(Intercept) 22.33602918
crim        -0.98873050
zn           1.01188826
indus        .
chas         2.84483528
nox         -2.03874738
rm           2.60993299
age          0.01792989
dis         -3.05132205
rad          2.20980293
tax         -1.85501217
ptratio     -1.99372747
lstat       -3.90427161
```

```
selected_vars
```

```
 [1] "crim"    "zn"      "chas"    "nox"     "rm"      "age"     "dis"
 [8] "rad"     "tax"     "ptratio" "lstat"
```

- sparse matrix
- these values are being calculated using gradient descent
- the values that have a dot are '0'

    1. the final model is saying that we should have a model that drops age and indus (these were the 2 variables that stepwise regression told us to drop)

```
full_model <- lm(medv ~ ., data=df)
lasso_model <- lm(y ~ X[, selected_vars])
```

```
summary(lasso_model)
```

```
Call:
lm(formula = y ~ X[, selected_vars])

Residuals:
     Min       1Q   Median       3Q      Max 
-15.1267  -2.7487  -0.5902   1.9056  26.2609 

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)    
(Intercept)                22.3350     0.2213 100.914  < 2e-16 ***
X[, selected_vars]crim     -1.0462     0.2834  -3.691 0.000248 ***
X[, selected_vars]zn        1.0878     0.3215   3.383 0.000773 ***
X[, selected_vars]chas      2.8591     0.8647   3.307 0.001013 ** 
X[, selected_vars]nox      -2.1478     0.4296  -4.999 8.01e-07 ***
X[, selected_vars]rm        2.5646     0.2938   8.728  < 2e-16 ***
X[, selected_vars]age       0.1016     0.3748   0.271 0.786563    
X[, selected_vars]dis      -3.1585     0.4146  -7.617 1.33e-13 ***
X[, selected_vars]rad       2.4850     0.5593   4.443 1.09e-05 ***
X[, selected_vars]tax      -2.0764     0.5749  -3.611 0.000336 ***
X[, selected_vars]ptratio  -2.0217     0.2836  -7.130 3.59e-12 ***
X[, selected_vars]lstat    -3.9355     0.3602 -10.927  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.793 on 494 degrees of freedom
Multiple R-squared:  0.7343,	Adjusted R-squared:  0.7284 
F-statistic: 124.1 on 11 and 494 DF,  p-value: < 2.2e-16
```

LASSO summary

- Lasso is useful because it is one step
- In the lasso model, in order to select an appropriate model, need to create model, looking at mean square error and choosing lambda value that is appropriate
- Variable selection has finite (set) amount of steps
- lasso is more efficient for data sets with TONS of variables

Gradient descent

- Used for solving one of the penalized estimators problems
- General recipe for fitting models
- Derivative is telling us slope (for small change in x, what is change in y)

- If you end up with a minimum point, the derivative will be flat (slope = 0, no change in y for change in x)
- A minimizer is characterized by 2 points
    1. derivative has slope of 0
    2. the 2nd derivative has to be positive
- To do gradient descent, compute derivative with respect to every parameter (partial derivative)

Recall that the solution to a regression problem is given by

$$(b_1, b_2, \dots, b_p) = \underset{\beta_1 \dots \beta_p}{\arg\min} L(\beta_0, \beta_1, \dots, \beta_p)$$

where $L(\beta_0, \beta_2, \dots, \beta_p)$ is referred to as the loss function. If we want to find the values of $(\beta_0, \beta_2, \dots, \beta_p)$ which minimize $L()$, then using the general principle from calculus, we are interested in looking for values such that the partial derivative with respect to each β is 0.

In the case of linear regression, the derivatives can be computed by hand, and there exists a closed form solution to the above system of equations

However, in many other models, we don't have a method for obtaining closed form solutions. In such cases, the general strategy is as follows:

1. Compute gradient
2. Choose a step size η between (0,1)
    - Start off at some randomized initialized value and at every step, choose a step size between 0 and 1
3. Perform gradient descent
    - Take one step in direction of negative gradient(direction that leads to decrease in the objective function, L)

- Repeat those steps until you reach some sort of stable minimum (when change of L is not significant to continue)

This is how lasso problem is being solved

```r
attach(cars)
```

Creating a loss function that calculates mean squared error

```r
Loss <- function(b,x,y) {
  squares <- (y - b[1] - b[2]*x)^2
  return(sum(squares))
}
b <- rnorm(2)
Loss(b, cars$speed, cars$dist)
```

```
[1] 119038.4
```

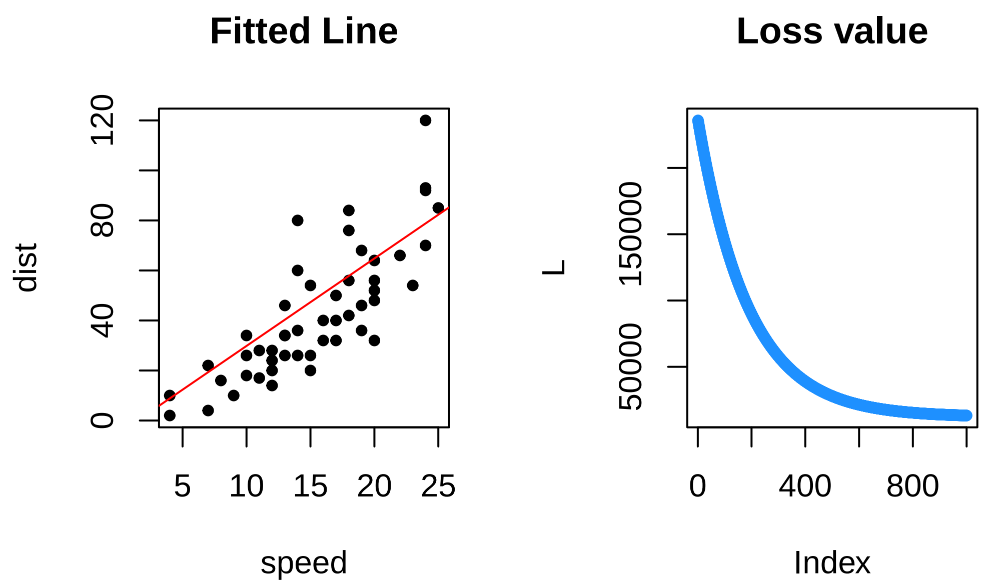
```r
# define a function to compute the gradients
grad <- function(b, Loss, x,y, eps=1e-5){
  b0_up <- Loss(c(b[1]+eps, b[2]),x,y)
  b0_dn <- Loss(c(b[1]-eps, b[2]),x,y)

  b1_up <- Loss(c(b[1], b[2]+eps),x,y)
  b1_dn <- Loss(c(b[1], b[2]-eps),x,y)

  grad_b0_L <- (b0_up - b0_dn) / (2 * eps)
  grad_b1_L <- (b1_up - b1_dn) / (2 * eps)

  return(c(grad_b0_L, grad_b1_L))
}

grad(b,Loss, cars$speed, cars$dist)
```

```
[1]  -4196.157 -74996.806
```

```r
steps <- 1000
L <- rep(Inf, steps)
eta <- 1e-7
b <- 10 * rnorm(2)

for (i in 1:steps){
  b <- b - eta * grad(b, Loss, cars$speed, cars$dist)
  L[i] <- Loss(b, cars$speed, cars$dist)
}
```

Creates a plot that shows the loss value for each index compared to the fitted line for the variables we plotted

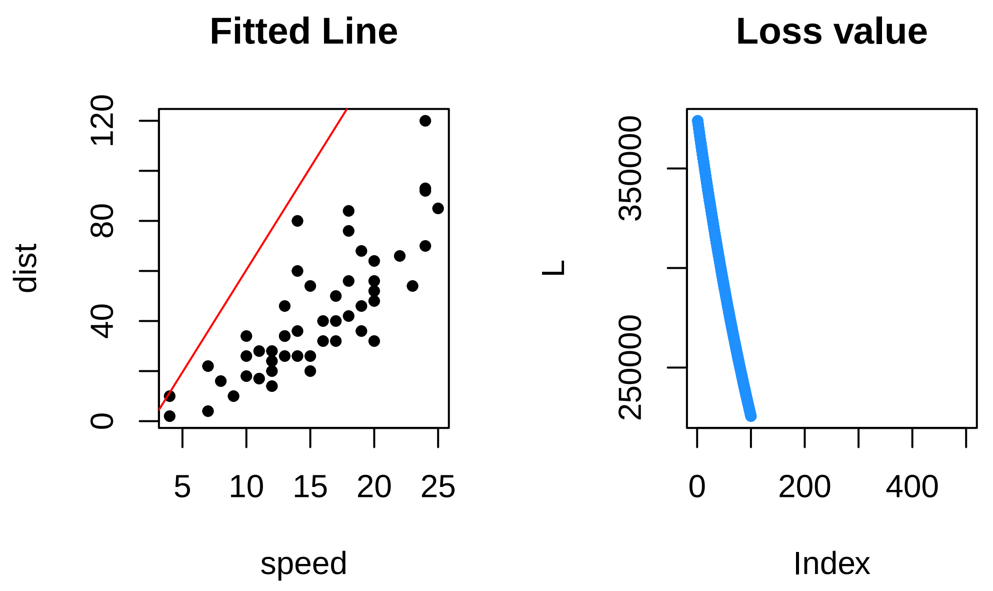
```r
options(repr.plot.width=12, repr.plot.height=7)
par(mfrow=c(1,2))
# Plot the final result
plot(dist ~ speed, cars, pch=20, main = "Fitted Line")
abline(b, col = 'red')

# Plot the change in loss function value
plot(L, type ='b', pch=20, col='dodgerblue', main='Loss value')
```

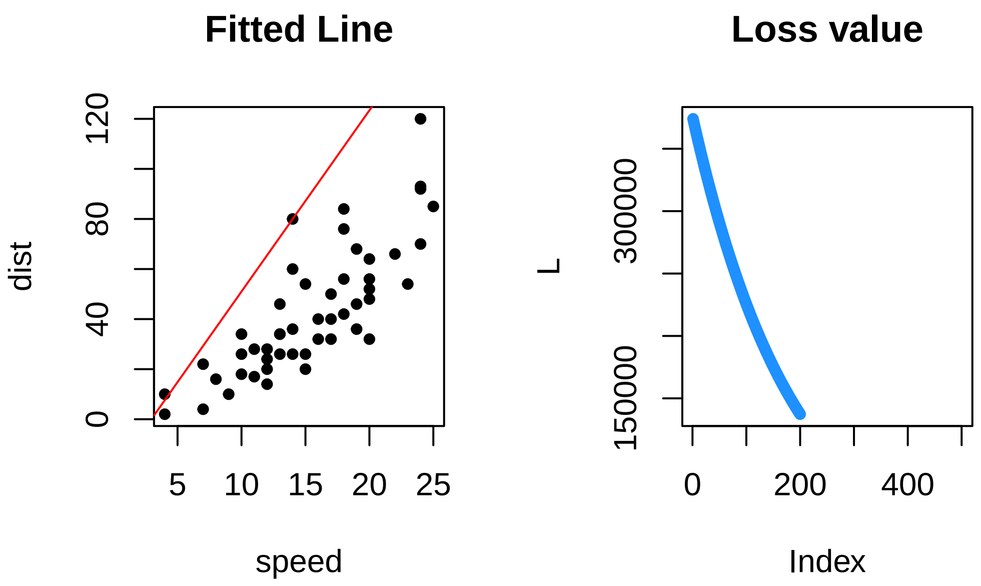


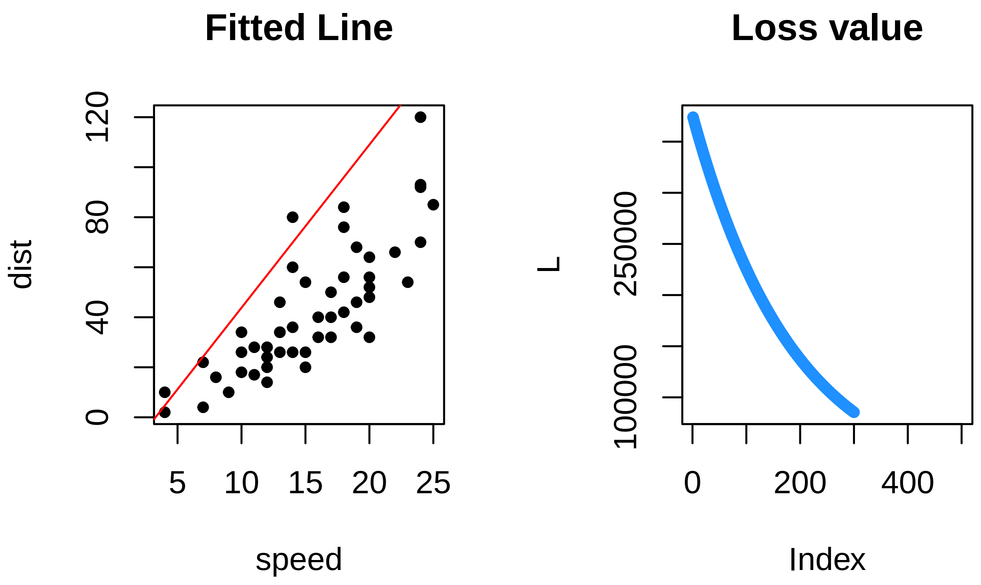
This next code chunk breaks down the loss function into various parts so you can see how the loss function progress at given indexes, along with the associated fitted line for the distance and speed plot

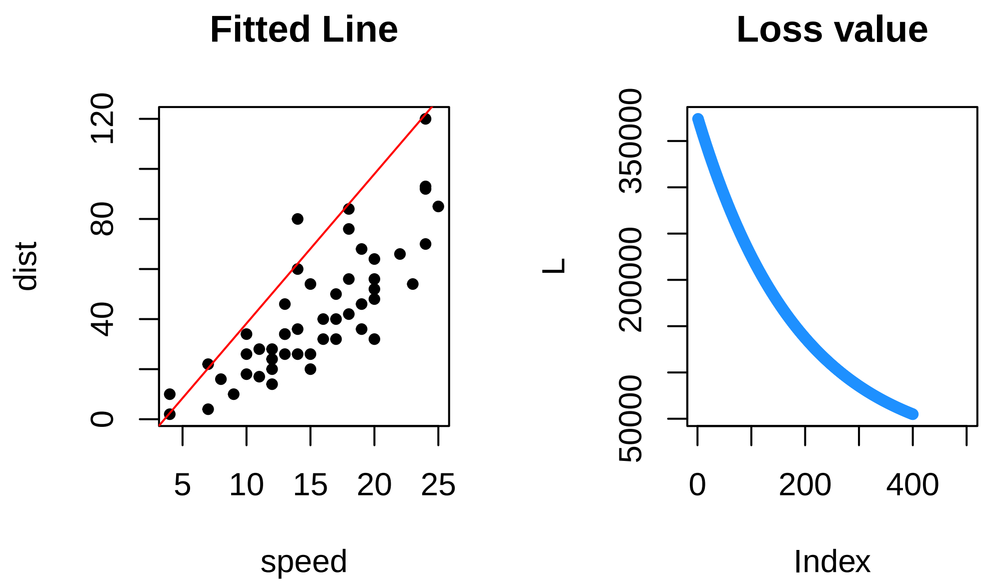
```r
options(repr.plot.width=12, repr.plot.height=7)
steps <- 500
L <- rep(Inf, steps)
eta <- 1e-7
b <- 10 * rnorm(2)

for (i in 1:steps){
  b <- b - eta * grad(b, Loss, cars$speed, cars$dist)
```

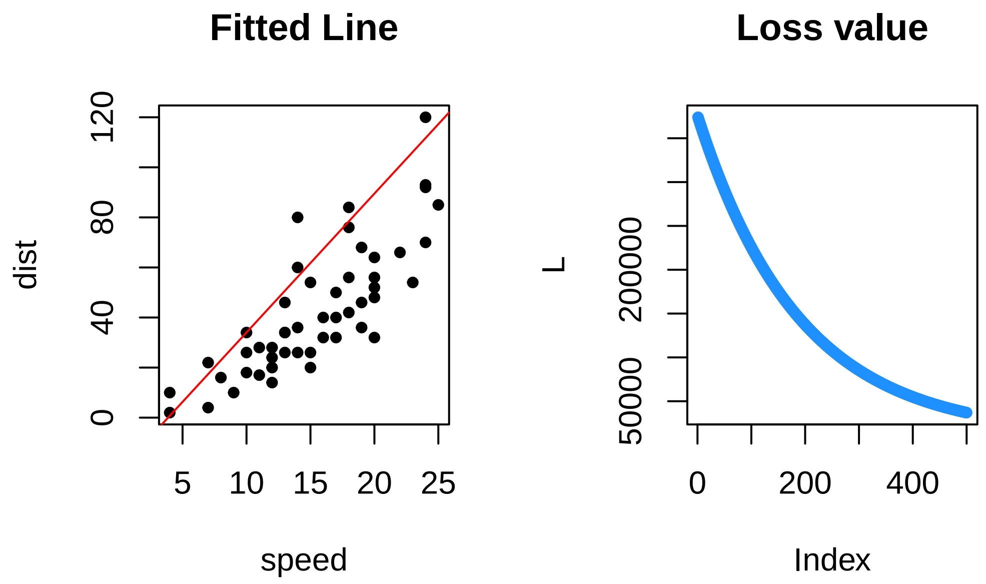
```
    L[i] <- Loss(b, cars$speed, cars$dist)

    if (i %% 100 == 0){
      par(mfrow=c(1,2))
      # Plot the final result
      plot(dist ~ speed, cars, pch=20, main = "Fitted Line")
      abline(b, col = 'red')

      # Plot the change in loss function value
      plot(L, type ='b', pch=20, col='dodgerblue', main='Loss value')
    }
  }
```




Fitted Line
Loss value
dist
0
40
80
120
5
10
15
20
25
speed
L
150000
300000
0
200
400
Index


Fitted Line
Loss value
dist
0
40
80
120
5
10
15
20
25
speed
L
100000
250000
0
200
400
Index


Fitted Line
dist
0
40
80
120
5
10
15
20
25
speed
Loss value
L
50000
200000
350000
0
200
400
Index


Fitted Line
dist
0
40
80
120
5
10
15
20
25
speed
Loss value
L
50000
200000
0
200
400
Index

Thursday, Feb 23

❗ TIL

Include a *very brief* summary of what you learnt in this class here.
Today, I learnt the following concepts in class:

1. Automatic differentiation
2. Cross validation
3. k-fold Cross Validation

Automatic differentiation

- Get rid of functions that are long/tedious to write out (ex. the gradient descent function we wrote before) and numerical instability
- Want to be able to write out loss function & automatically be able to calculate loss for each parameter
- Automatic differentiation helps calculate gradients for any function without the need to solve tedious calculus problems

```r
# vector of 5 values
# c(5,1) tells shape --> 5 rows, 1 column
# 2nd part says that it's matrix, so you can calculate the gradient descent
x <- torch_randn(c(5,1), requires_grad = TRUE)
x
```

```
torch_tensor
-0.7296
-1.5856
-0.0168
 0.9341
 0.1378
[ CPUFloatType{5,1} ][ requires_grad = TRUE ]
```

- matrix = 2D tensor
- vector = 1D tensor

```r
# sqrt(sum(as_array(x)^2)^10 is what torch_norm does
f <- function(x){
  torch_norm(x)^10
}
```

```
y <- f(x)
y
```

```
torch_tensor
947.542
[ CPUFloatType{} ][ grad_fn = <PowBackward0> ]
```

```
# this stops compiler from keeping track of changes to x & start computing gradients
y$backward()
```

```
x$grad
```

```
torch_tensor
-1755.4103
-3814.8384
  -40.3345
 2247.4229
  331.6463
[ CPUFloatType{5,1} ]
```

```
(5*torch_norm(x)^8) * (2*x)
```

```
torch_tensor
-1755.4103
-3814.8384
  -40.3345
 2247.4229
  331.6463
[ CPUFloatType{5,1} ][ grad_fn = <MulBackward0> ]
```

```
x <- torch_randn(c(10,1), requires_grad = TRUE)
x
```

```
torch_tensor
 0.8346
 1.5741
```

```
-0.5107
-0.8870
 0.2870
 0.5562
-0.3226
 0.9152
 0.4237
 1.2534
[ CPUFloatType{10,1} ][ requires_grad = TRUE ]
```

```r
y <- torch_randn(c(10,1), requires_grad = TRUE)
y
```

```
torch_tensor
 0.6836
-0.1887
-0.6461
-1.7009
 0.9143
 1.8650
 0.1154
 0.4989
 0.0260
-0.7580
[ CPUFloatType{10,1} ][ requires_grad = TRUE ]
```

```r
f <- function(x,y) {
  sum(x*y)
}

z <- f(x,y)
z
```

```
torch_tensor
2.89232
[ CPUFloatType{} ][ grad_fn = <SumBackward0> ]
```

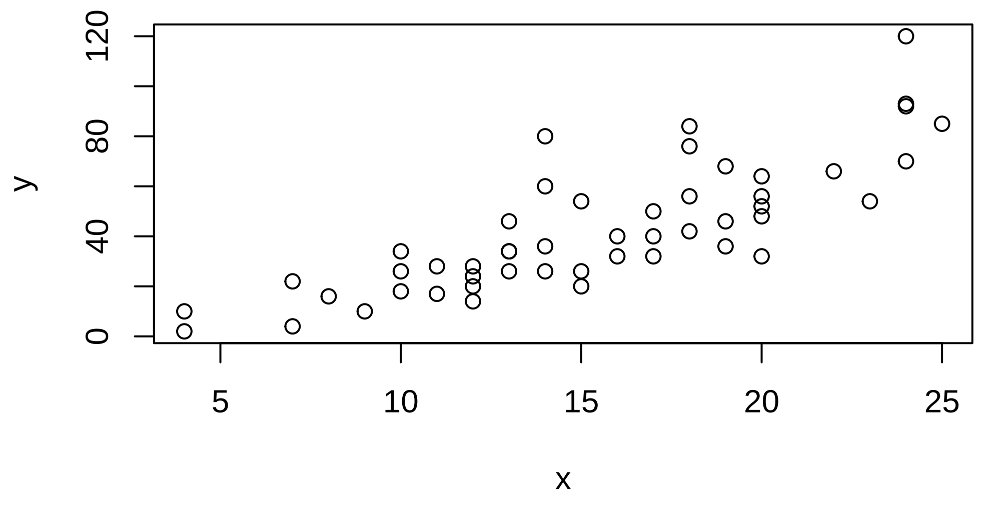
```r
z$backward()
```

```r
c(x$grad, y$grad)
```

```
[[1]]
torch_tensor
 0.6836
-0.1887
-0.6461
-1.7009
 0.9143
 1.8650
 0.1154
 0.4989
 0.0260
-0.7580
[ CPUFloatType{10,1} ]

[[2]]
torch_tensor
 0.8346
 1.5741
-0.5107
-0.8870
 0.2870
 0.5562
-0.3226
 0.9152
 0.4237
 1.2534
[ CPUFloatType{10,1} ]
```

Example of automatic differentiation using the cars data set

```r
# using the speed and distance variables
x <- torch_tensor(cars$speed, dtype = torch_float())
y <- torch_tensor(cars$dist, dtype = torch_float())

plot(x,y)
```



```
b <- torch_zeros(c(2,1), dtype=torch_float(), requires_grad= TRUE)
b
```

```
torch_tensor
 0
 0
[ CPUFloatType{2,1} ][ requires_grad = TRUE ]
```

```
loss <- nn_mse_loss()
```

```
b <- torch_zeros(c(2,1), dtype=torch_float(), requires_grad = TRUE)
steps <- 5000
L <- rep(Inf, steps)
eta <- 0.5
optimizer <- optim_adam(b, lr=eta)


# boiler plate for any optimization that we do
for (i in 1:steps){
  # compute predicted value (contains slope and intercept)
```

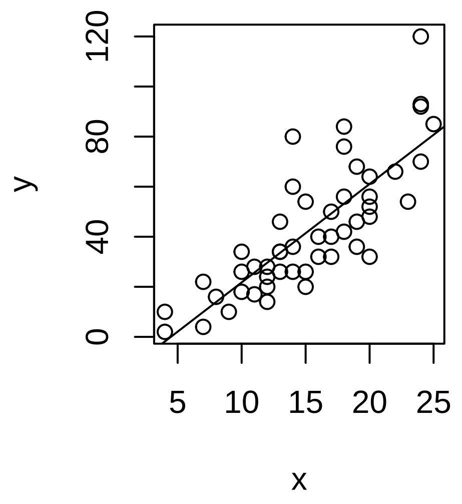
```
    y_hat <- x * b[2] + b[1]
    # compute loss l (want to compute gradient with respect to loss)
    l <- loss(y_hat,y)

    L[i] <- l$item()
    optimizer$zero_grad()
    # tells to stop here and take gradient from here
    l$backward()
    # tells to take step in direction of negative gradient for thing inside optimizer
    optimizer$step() # more intelligent optimizer than previous formula used

    if(i %in% c(1:10) || i %% 200 == 0){
      cat(sprintf("Iteration: %s\t Loss value: %s\n", i, L[i]))
    }
  }
```

```
Iteration: 1     Loss value: 2498.06005859375
Iteration: 2     Loss value: 1759.53002929688
Iteration: 3     Loss value: 1174.45300292969
Iteration: 4     Loss value: 742.353759765625
Iteration: 5     Loss value: 457.703643798828
Iteration: 6     Loss value: 307.684936523438
Iteration: 7     Loss value: 270.263397216797
Iteration: 8     Loss value: 314.067993164062
Iteration: 9     Loss value: 401.761566162109
Iteration: 10    Loss value: 496.908325195312
Iteration: 200   Loss value: 231.474166870117
Iteration: 400   Loss value: 227.114730834961
Iteration: 600   Loss value: 227.070495605469
Iteration: 800   Loss value: 227.070404052734
Iteration: 1000  Loss value: 227.070404052734
Iteration: 1200  Loss value: 227.070404052734
Iteration: 1400  Loss value: 227.070404052734
Iteration: 1600  Loss value: 227.070404052734
Iteration: 1800  Loss value: 227.070404052734
Iteration: 2000  Loss value: 227.070404052734
Iteration: 2200  Loss value: 227.070404052734
Iteration: 2400  Loss value: 227.070434570312
Iteration: 2600  Loss value: 227.070434570312
Iteration: 2800  Loss value: 227.070434570312
Iteration: 3000  Loss value: 227.070434570312
Iteration: 3200  Loss value: 227.070434570312
```

```
Iteration: 3400  Loss value: 227.070388793945
Iteration: 3600  Loss value: 227.070404052734
Iteration: 3800  Loss value: 227.070434570312
Iteration: 4000  Loss value: 227.070404052734
Iteration: 4200  Loss value: 227.070434570312
Iteration: 4400  Loss value: 227.070434570312
Iteration: 4600  Loss value: 227.070434570312
Iteration: 4800  Loss value: 227.070404052734
Iteration: 5000  Loss value: 227.070404052734
```

- Brings the loss down on a much quicker trajectory

```r
options(repr.plot.width = 12, repr.plot.height = 7)

par(mfrow=c(1,2))
plot(x,y)

abline(as_array(b))
```



Cross Validation

```
df <- Boston %>% drop_na()
head(df)
```

```
     crim zn indus chas   nox    rm  age    dis rad tax ptratio lstat medv
1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3  4.98 24.0
2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8  9.14 21.6
3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8  4.03 34.7
4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7  2.94 33.4
5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7  5.33 36.2
6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7  5.21 28.7
```

```
dim(df)
```

```
[1] 506  13
```

Spliting data into training (80%) and testing sets (20%)

```
k <- 5
fold <- sample(1:nrow(df), nrow(df)/k)
fold
```

```
  [1] 156 233  54 239 468  32 155  60 292 255  85 312 214 504 254 442  74  80
 [19]  91 208 482 153 194  34 396  52 415  50 367 169 160 328 443 181 414 488
 [37] 379 276 472 248 401 265 308 463 115 491 326 100 161  89  72 219 133 282
 [55] 431 256 373   8 118  67 433 164   4 253 172 300 249 283 501 475  26 170
 [73] 439 259 111 263  81 321 425 389 484 336  90 217  29 458 323 268 307 130
 [91] 392 429 306 145 502  82   7  73 377  77 424
```

- AIC is a goodness of fit parameter (similar to R^2)
- only creating model using training data
- use parameters from that model to predict what the values would be on test set
- see the discrepancy between predicted value and actual error (test error)

```
train <- df %>% slice(-fold)
test <- df %>% slice(fold)
```

```r
model <- lm(medv ~., data = train)
summary(model)
```

```

Call:
lm(formula = medv ~ ., data = train)

Residuals:
     Min       1Q   Median       3Q      Max 
-13.6546  -2.7277  -0.5252   1.8130  24.8137 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  47.995487   5.468560   8.777  < 2e-16 ***
crim         -0.136508   0.034577  -3.948 9.35e-05 ***
zn            0.056832   0.015157   3.750 0.000204 ***
indus         0.022693   0.066530   0.341 0.733216    
chas          2.468396   0.991332   2.490 0.013188 *  
nox         -20.712776   4.434589  -4.671 4.13e-06 ***
rm            3.027981   0.469045   6.456 3.19e-10 ***
age           0.010074   0.014917   0.675 0.499847    
dis          -1.677714   0.223380  -7.511 4.02e-13 ***
rad           0.277207   0.073249   3.784 0.000178 ***
tax          -0.012013   0.004036  -2.976 0.003098 ** 
ptratio      -0.973225   0.147516  -6.597 1.36e-10 ***
lstat        -0.603428   0.055212 -10.929  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.771 on 392 degrees of freedom
Multiple R-squared:  0.7359,	Adjusted R-squared:  0.7278 
F-statistic: 91.01 on 12 and 392 DF,  p-value: < 2.2e-16
```

```r
y_test <- predict(model, newdata = test)
```

```r
# mean squared prediction error
mspe <- mean((test$medv - y_test)^2)
mspe
```

```
[1] 24.9762
```

- If you make training/testing 50-50, then the mspe will decrease/increase??

    1. This depends on the portion of data that is selected in the 50% training set

- To get rid of variability, use "k-fold cross validation"

k-Fold Cross Validation

- uses similar logic as before but now you pick number of folds
- split data into k disjoint subsets of rows

    1. 1000 rows becomes k datasets of 1000/k rows

- then you select 1 of the 5 datasets as test, and rest as training set
- train on 4, predict on test and make mspe
- do this for all 5 blocks, using each as test
- have a mspe for every fold (in this case have 5 mspe's)
- find average of those mspe

```r
k <- 5
folds <- sample(1:k, nrow(df), replace = T)
```

```r
# function fo creating training sets for 5 folds of the data set
df_folds <- list()

for (i in 1:k){
  df_folds[[i]] <- list()
  df_folds[[i]]$train = df
}
```