Stats216v: Statistical Learning

Stanford University
Summer 2017
Gyu-Ho Lee (gyuhox@gmail.com (mailto:gyuhox@gmail.com))

Q1.

Suppose you have collected a dataset of information about 40 patients, ages 55 to 60 years old. You have variables X_1 = weight, X_2 = years they have smoked, and Y = whether or not they have had a heart attack. You fit a logistic regression model, which yields the following coefficients: $\hat{\beta}_0 = -3.8$, $\hat{\beta}_1 = 0.007$, $\hat{\beta}_2 = 0.03$.

(a) Estimate the probability that a new patient who weighs 143 lbs and has smoked for 17 years will have a heart attack.

Gyu-Ho's Answer: 0.09203808291997737.

Logistic regression would be in $P(X) = \frac{e^{\beta_0+\beta_1*X_1+\beta_2*X_2}}{1+e^{\beta_0+\beta_1*X_1+\beta_2*X_2}} = \frac{e^{-3.8+0.007*143+0.03*17}}{1+e^{-3.8+0.007*143+0.03*17}} = 0.09203808291997737$

(b) Suppose a patient does not smoke: X_2 = 0. At what weight would we predict the chance of having a heart attack is 10%?

Gyu-Ho's Answer: 228.96791752339718.

$\frac{e^{-3.8+0.007*X_1}}{1+e^{-3.8+0.007*X_1}} = \frac{e^{-3.8+0.007*228.96791752339718}}{1+e^{-3.8+0.007*228.96791752339718}} = 0.09999999999999999$

Q2.

Recall that the Lasso estimate for β is the minimizer of the following expression:

$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}|\beta_j|$

where λ is a tuning parameter.

(a) We call $\lambda \sum_{j=1}^{p}|\beta_j|$ the Lasso penalty term. Why do we call it a penalty?

Gyu-Ho's Answer: Like ridge regression, lasso aims to shrink coefficient estimates towards zero, thus minimize the penalty $\lambda \sum_{j=1}^{p} |\beta_j|$. Lasso is a shrinking method, so it needs impose a penalty on the complexity of model, in that the bigger the penalty is, the less coefficients are zeros, the more predictors are, thus more complex.

The model pays a penalty of $\lambda|\beta_j|$ for including term β_j in the model. This discourages the sum of the magnitude of the coefficients from being large. In other words it favors models with small sum of the coefficients.

(b) In terms of other techniques from Stats 216v, what value of β gives the minimum of the above expression when $\lambda = 0$?

Gyu-Ho's Answer: When $\lambda = 0$, lasso returns the least squares fit solution.

The OLS estimate $\hat{\beta}_{OlS}$.

(c) What value of β gives the minimum when $\lambda = \infty$?

Gyu-Ho's Answer: When $\lambda = \infty$, lasso returns the null model where all coefficient estimates are 0 ($\beta = 0$).

All the coefficients become zero in this case.

(d) Briefly explain that what will happen to each of the following quantities as we increase the parameter λ from 0 to ∞:

i. The training RSS.

Gyu-Ho's Answer: Steadily increase. Increasing λ fits simpler models, less flexible to fit exact training data, thus increased training RSS.

The training RSS will increase steadily since we are going from a very flexible model to a model that can just capture the mean.

ii. The test RSS.

Gyu-Ho's Answer: Decrease initially as we are not overfitting over training data. And then eventually start increasing in a U shape when the model becomes too simple to capture the true model, thus test RSS goes up.

Initially, with $\lambda = 0$, the model starts with the OLS estimates, which minimize the sum of the squared residuals for the training data, and so generally does not perform so well on the test data, which results in relatively high RSS. As λ is increased, the values of the coefficients are reduced and so is the overfitting, which lessens the RSS. Eventually, as the coefficients approach zero, the model loses too much of its flexibility and the test RSS increases.

iii. The variance of the β_i's.

Gyu-Ho's Answer: Steadily decrease, as increasing λ fits simpler models, reducing the variance.

When $\lambda = 0$ we have the OLS estimates. As λ increases, the model becomes simpler which results in less variance for the coefficients.

iv. The bias.

Gyu-Ho's Answer: Steadily increase, as increasing λ fits simpler models, reducing the variance, thus bigger squared bias.

We start with the OLS estimates, and as λ grows the model loses flexibility by shrinking the coefficients to zero. As a result, the model fits less accurately to the training data, and so the bias increases.

In [22]:
```
LoadLibraries = function() {
    library(MASS)
    install.packages("ISLR")
    library(ISLR)
    install.packages("leaps")
    library(boot)
    library(class)
    library(glmnet)
    library(leaps)
    install.packages("pls")
    library(pls)
    print("Libraries have been loaded!")
}

LoadLibraries()
```

```
Updating HTML index of packages in '.Library'
Making 'packages.html' ... done
Updating HTML index of packages in '.Library'
Making 'packages.html' ... done
Updating HTML index of packages in '.Library'
Making 'packages.html' ... done

[1] "Libraries have been loaded!"
```

Q3.

The `Default` data set in the `ISLR` library contains data about 10000 simulated credit card customers. Our goal is to model the probability that a customer will default on their debt. We will use a logistic regression model using `default` as the response and income and `balance` as features. For this problem, we are interested in the standard errors of our estimates of the logistic regression coefficients. We will compute standard errors in two ways: (1) using the bootstrap and (2) using the standard formula for computing standard errors, which is implemented in the `glm()` function.

(a) In two sentences or less, explain what the “standard error” for a coefficient in the model means.

Gyu-Ho's Answer: Estimated variability of a coefficient that indicates the precision of coefficient estimates.

Due to the randomness in the data, the coefficient of a feature (for example, β_{income}) is a random quantity. The standard deviation of this random quantity is known as the standard error.

(b) Load the data into your workspace using the following commands:

```
library("ISLR")
data("Default")
```

Next, using the `summary()` and `glm()` functions, determine the estimated standard errors for the coefficients associated with income and balance in the multiple logistic regression model.

In [23]:
```
# names(Default)
# 'default' 'student' 'balance' 'income'

# logistic regression model using default as the response and income and
 balance as features
glm.fit = glm(default~income+balance, data=Default, family=binomial)
summary(glm.fit)
```

```
Call:
glm(formula = default ~ income + balance, family = binomial,
    data = Default)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4725  -0.1444  -0.0574  -0.0211   3.7245

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2920.6  on 9999  degrees of freedom
Residual deviance: 1579.0  on 9997  degrees of freedom
AIC: 1585

Number of Fisher Scoring iterations: 8
```

The estimated standard error for income is 4.99e-06 and the estimated standard error for balance is 2.27e-04.

(c) Write a function `boot_fn()` that takes as input the `Default` data set as well as an index of the observations, and that outputs the coefficient estimates for `income` and `balance` in the multiple logistic regression model.

In [24]:
```
summary(glm.fit)$coefficients

# estimated standard errors for the coefficients associated with income
 and balance
summary(glm.fit)$coefficients[,2]
```

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-1.154047e+01	4.347564e-01	-26.544680	2.958355e-155
income	2.080898e-05	4.985167e-06	4.174178	2.990638e-05
balance	5.647103e-03	2.273731e-04	24.836280	3.638120e-136

(Intercept)	0.434756357486525
income	4.98516717786299e-06
balance	0.000227373141926647

Gyu-Ho's Answer: Estimated standard errors for the coefficients are `0.434756357486525` for intercept β_0, `4.98516717786299e-06` for `income` β_1, `0.000227373141926647` for `balance` β_2.

(d) Use the `boot()` function together with your `boot_fn()` to estimate the standard errors of the logistic regression coefficients for `income` and `balance` in the multiple logistic regression model. Do 100 bootstrap replications, using `set.seed(2017)` to set the seed.

In [25]:
```
# estimate the standard errors of the logistic regression coefficients
# for income and balance in the multiple logistic regression model
boot_fn = function(data, index) {
    fit = glm(default~income+balance, data=data, family=binomial,
subset=index)
    return (coef(fit))
}

set.seed(2017)
boot.fit = boot(Default, boot_fn, 100)
```

In [26]:
```
boot.fit
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Default, statistic = boot_fn, R = 100)


Bootstrap Statistics :
         original        bias     std. error
t1* -1.154047e+01  1.024721e-02 4.824698e-01
t2*  2.080898e-05 -1.103654e-06 4.948009e-06
t3*  5.647103e-03  2.076323e-05 2.643337e-04
```

Gyu-Ho's Answer: Estimated the standard errors of the logistic regression coefficients are `4.824698e-01` for intercept β_0, `4.948009e-06` for `income` β_1, `2.643337e-04` for `balance` β_2.

The estimated standard error for income is 4.95e-06 and the estimated standard error for balance is 2.64e-04.

(e) Comment on the estimated standard errors from the two approaches. Using the standard errors from `glm()`, which of the two predictors are statistically significant?

Gyu-Ho's Answer: Estimated standard errors from two methods are very close. Based on the standard errors from `glm()`, the predictor `balance` is more statistically significant because z-statistic of `balance` is large, thus providing evidence against the null hypothesis.

The estimated standard errors for R income are remarkably close. The estimated standard errors for the intercept and balance are reasonably close, coming within about 20%.

Q4.

This question should be answered using the `Weekly` dataset, which is part of the `ISLR` package. This data is similar in nature to the `SMarket` data used in section 4.6 of out textbook, except that it contains 1,089 weekly returns for 21 years, from the beginning of 1990 to the end of 2010.

(a) Use the full dataset to perform a logistic regression with `Direction` as the response and the five lag variables plus `Volume` as predictors. Call your model `glm.fit`. Use the summary function to print the results. Do any of the predictors appear to be statistically significant? If so, which ones?

In [27]:
```
# names(Weekly)
# 'Year' 'Lag1' 'Lag2' 'Lag3' 'Lag4' 'Lag5' 'Volume' 'Today' 'Direction'

# logistic regression with Direction as the response and the five lag va
riables plus Volume as predictors
glm.fit = glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume, data=Weekly, fa
mily=binomial)
summary(glm.fit)
```

```
Call:
glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
    Volume, family = binomial, data = Weekly)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.6949  -1.2565   0.9913   1.0849   1.4579

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.26686    0.08593   3.106   0.0019 **
Lag1        -0.04127    0.02641  -1.563   0.1181
Lag2         0.05844    0.02686   2.175   0.0296 *
Lag3        -0.01606    0.02666  -0.602   0.5469
Lag4        -0.02779    0.02646  -1.050   0.2937
Lag5        -0.01447    0.02638  -0.549   0.5833
Volume      -0.02274    0.03690  -0.616   0.5377
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1496.2  on 1088  degrees of freedom
Residual deviance: 1486.4  on 1082  degrees of freedom
AIC: 1500.4

Number of Fisher Scoring iterations: 4
```

Gyu-Ho's Answer: Predictors of `Lag2` and `Lag5` and `Volume` appear to be statistically significant since they have relatively larger z-statistic values.

Lag 2 and the intercept appear to be statistically significant.

(b) Use the following code to produce the confusion matrix for this problem.

```
glm.probs = predict(glm.fit, type="response")
glm.pred = rep("Down", length(glm.probs))
glm.pred[glm.probs > 0.5] = "Up"
table(glm.pred, Weekly$Direction)
```

Explain what the confusion matrix is telling you about the types of mistakes made by logistic regression.

In [28]:
```
glm.probs = predict(glm.fit, type="response")
glm.pred = rep("Down", length(glm.probs))
glm.pred[glm.probs > 0.5] = "Up"

contrasts(Weekly$Direction)
table(glm.pred, Weekly$Direction)
```

	Up
Down	0
Up	1

```
glm.pred Down  Up
    Down   54  48
    Up    430 557
```

In [29]:
```
# fraction of days where the direction was wrong
# (48+430) / (54+48+430+557)
mean(glm.pred != Weekly$Direction)
```

0.438934802571166

Gyu-Ho's Answer: The confusion matrix shows that out of total 1,089 predictions, 48 are predicted as "Down" but actual direction was "Up" (False "Down"s), 430 are predcticed as "Up" but actually direction was "Down" (False "Up"s).

When the model predicts ‘down’ then the prediction was correct 54/102 times. When the model predicts ‘up’ the prediction was correct 557/ 987 times.

(c) Now fit the logistic regression model using a training data period from 1990 to 2007, with `Lag1`, `Lag2`, and `Lag3` as the only predictors. Compute the confusion matrix and the overall fraction of correct predictions for the held out data (that is, the data from 2008 and 2010).

In [30]:
```
# training data period from 1990 to 2007
train = (Weekly$Year <= 2007)

# data from 2008 and 2010
Weekly.test = Weekly[!train,]

# logistic regression model using training data with Lag1, Lag2, Lag3
logit.fit = glm(Direction~Lag1+Lag2+Lag3, data=Weekly, family=binomial,
subset=train)
summary(logit.fit)

# compute overall fraction of correct predictions for the held out data
 (test data)
glm.probs = predict(logit.fit, Weekly.test, type="response")
glm.pred = rep("Down", nrow(Weekly.test))
glm.pred[glm.probs > 0.5] = "Up"

# confusion matrix in test data
table(glm.pred, Weekly.test$Direction)

# fraction of correct predictions in test data
mean(glm.pred == Weekly.test$Direction)
```

```
Call:
glm(formula = Direction ~ Lag1 + Lag2 + Lag3, family = binomial,
    data = Weekly, subset = train)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-1.457  -1.267   1.012   1.080   1.410

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.23889    0.06698   3.567 0.000361 ***
Lag1        -0.04643    0.03256  -1.426 0.153810
Lag2         0.04232    0.03259   1.298 0.194123
Lag3        -0.01370    0.03235  -0.423 0.671932
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1280.6  on 932  degrees of freedom
Residual deviance: 1276.2  on 929  degrees of freedom
AIC: 1284.2

Number of Fisher Scoring iterations: 4


glm.pred Down Up
    Down   10  9
    Up     62 75
```

0.544871794871795

Gyu-Ho's Answer: Overall fraction of correct predictions for the held out data is 0.544871794871795.

When the model predicts ‘down’ then the prediction was correct 10/19 times. When the model predicts ‘up’ the prediction was correct 75/137 times. This is a mis-classification rate of .455.

(d) Repeat (c) using `LDA`. Use `library(MASS)` to work with the `lda()` command.

In [31]:
```
# training data period from 1990 to 2007
train = (Weekly$Year <= 2007)

# data from 2008 and 2010
Weekly.test = Weekly[!train,]

# linear discriminant analysis using training data with Lag1, Lag2, Lag3
lda.fit = lda(Direction~Lag1+Lag2+Lag3, data=Weekly, family=binomial, su
bset=train)
lda.fit

# compute overall fraction of correct predictions for the held out data
 (test data)
lda.pred = predict(lda.fit, Weekly.test, type="response")

# confusion matrix in test data
table(lda.pred$class, Weekly.test$Direction)

# fraction of correct predictions in test data
mean(lda.pred$class == Weekly.test$Direction)
```

```
Call:
lda(Direction ~ Lag1 + Lag2 + Lag3, data = Weekly, family = binomial,
    subset = train)

Prior probabilities of groups:
     Down        Up
0.4415863 0.5584137

Group means:
           Lag1       Lag2      Lag3
Down 0.29930825 0.07329612 0.2227597
Up   0.08858541 0.27110173 0.1431939

Coefficients of linear discriminants:
             LD1
Lag1 -0.33396602
Lag2  0.30485524
Lag3 -0.09792095


       Down Up
  Down   10  9
  Up     62 75
```

0.544871794871795

The confusion matrix is identical to the one from logistic regression. LDA and logistic regression have similar structure, so with this much data it is plausible that they ended up with the same predictions.

(e) Repeat (c) using `KNN` with `K = 1`. Invoke `library(class)` to work with the `knn()` command.

In [32]:
```
# training data period from 1990 to 2007
train = (Weekly$Year <= 2007)
train.X = cbind(Weekly$Lag1, Weekly$Lag2, Weekly$Lag3)[train,]
train.Direction = Weekly$Direction[train]

# data from 2008 and 2010
test.X = cbind(Weekly$Lag1, Weekly$Lag2, Weekly$Lag3)[!train,]

# KNN using training data with Lag1, Lag2, Lag3
set.seed(2017)
knn.pred = knn(train.X, test.X, train.Direction, k=1)

# confusion matrix in test data
table(knn.pred, Weekly.test$Direction)

# fraction of correct predictions in test data
mean(knn.pred == Weekly.test$Direction)
```

```
knn.pred Down Up
    Down   28 37
    Up     44 47
```

0.480769230769231

When the model predicts ‘down’ then the prediction was correct 28/65 times. When the model predicts ‘up’ the prediction was correct 47/91 times. This is a mis-classification rate of .519.

(f) Which of the models from parts (c), (d), and (e) appears to provide the best results on this data?

Gyu-Ho's Answer: (c) logistic regression and (d) LDA provide best results.

The LDA and logistic regression model have identical mis-classification rate, and it is lower than that of the knn model. The first two models appear to be doing a little bit better. Note that simply predicting ‘up’ every time would have better accuracy than the KNN model. This suggests that the KNN model is quite bad in this case.

(g) What is one scenario in which you might expect an LDA model to outperform a logistic regression model?

Gyu-Ho's Answer:

- When the classes are well-separated.
- When n is small and predictors are multivariate Gaussian (distribution of predictors are normal in each class).

LDA is assuming that the features X are generated from different normal distri- butions depending on the class of Y . When this is correct, it will do better. For example, imagine classifying two species of birds based on their wingspan. In this case, if we expect that the wingspans follow different normal distributions for each species, then LDA is probably better suited. Logistic regression can also suffer from numerical instabilities if the classes are perfectly separable, which doesn't happen with LDA.

(h) What is one scenario in which you might expect a `KNN` model to outperform a logistic regression model?

Gyu-Ho's Answer: When decision boundary is **highly** non-linear because KNN is a non-parametric approach.

KNN is a more flexible model, and may be able to fit nonlinear shapes that logistic regression cannot. Suppose that we are trying to predict whether a person is a college student based on their age. Given sufficient data, KNN would be able to predict 'yes' for people between 18 – 22 and 'no' for others. Logistic regression, which is a linear classifier, would always classify people below a certain age as 'yes' and everyone else as 'no', which would perform poorly in this setting.

Q5.

This question uses the `ALS` dataset from Homework 1. We saw that applying basic linear regression did not yield very good results, mainly because only a few of the many input predictors are actually strongly associated with the output. One alternative to remedy this problem is to use the Lasso, since it automatically does variable selection for us.

First, load the `als.RData` file.

In [38]:
```
loadedHW1 = load("hw-1-als.RData")
head(loadedHW1) # 'train.X' 'train.y' 'test.X' 'test.y'

rDataHW1 = get(loadedHW1)
head(rDataHW1)

dim(train.X)
head(train.y)
```

'train.X' 'train.y' 'test.X' 'test.y'

Onset.Delta	Symptom.Speech	Symptom.WEAKNESS	Site.of.Onset.Onset..Bulbar	Site.o
-341	1	1	1	0
-1768	0	1	0	1
-334	1	0	1	0
-268	0	1	0	1
-440	0	1	0	1
-773	0	1	0	1

1197 323

-0.922323232323232 -0.385274261603376 -0.118430609597925
-1.18584415584416 -1.01081180811808 -0.108702380952381

(a) Fit a regression model to the training data using the Lasso. Select the regularization parameter via cross-validation. To do so, you'll need to install (and then load) the package `glmnet`. Use the function `cv.glmnet()` within this package to fit the model with cross-validation. Before you do so, use `set.seed(2017)` right before running `cv.glmnet()` to ensure you get the same results as in the solutions. Store the result of `cv.glmnet()` in a variable called `lasso.cv`.

In [40]:
```
set.seed(2017)
lasso.cv = cv.glmnet(x=as.matrix(train.X), y=train.y)
names(lasso.cv$glmnet.fit)

# write.csv(lasso.cv$glmnet.fit, file="lasso.cv.csv")
# lasso.cv$glmnet.fit

bestlam = lasso.cv$lambda.min
bestlam

lasso.coef = predict(lasso.mod, type="coefficients", s=bestlam)
head(lasso.coef)
```

'a0' 'beta' 'df' 'dim' 'lambda' 'dev.ratio' 'nulldev' 'npasses' 'jerr' 'offset' 'call' 'nobs'

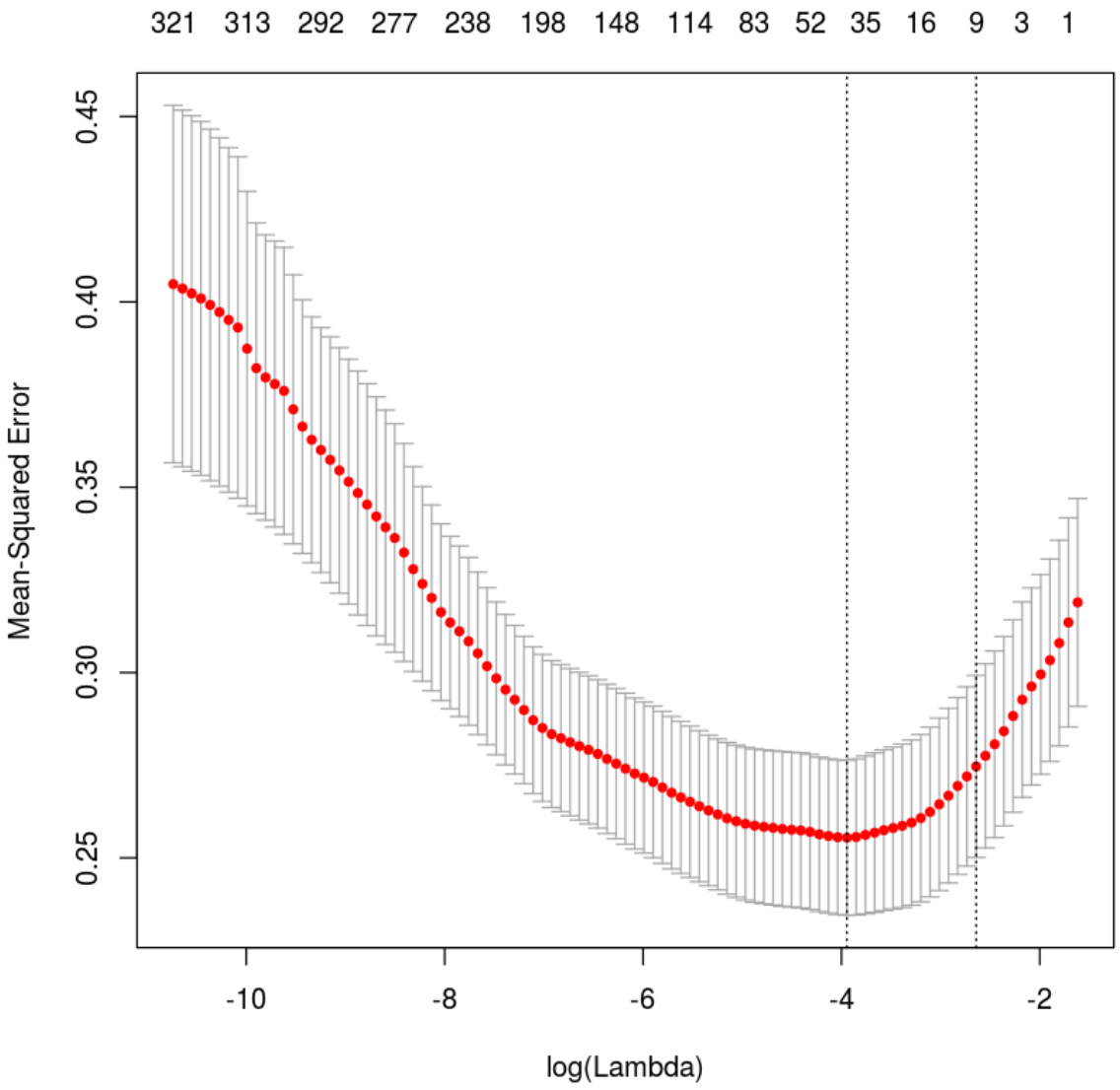
0.0193589194512196

```
6 x 1 sparse Matrix of class "dgCMatrix"
                                          1
(Intercept)                  -0.3832976029
(Intercept)                   .
Onset.Delta                  -0.0004004848
Symptom.Speech                .
Symptom.WEAKNESS              .
Site.of.Onset.Onset..Bulbar -0.0575485482
```

In [41]:
```
# wrong

# model.matrix to produce a matrix with X predictors
# also automatically transforms any qualitative variables into dummy var
iables
# x = model.matrix(train.y~., train.X)
# y = train.y

# grid of values ranging from λ=10^10 to λ=10^-2
# grid = 10^seq(10, -2, length=100)
# alpha=0 for ridge regression
# alpha=1 for lasso
# automatically standardize variables
# lasso.mod = glmnet(x, y, alpha=1, lambda=grid)
# lasso.mod = glmnet(x, y, alpha=1)

# set.seed(2017)
# lasso.cv = cv.glmnet(x, y, alpha=1)
```

(b) Produce a plot via `plot(lasso.cv)` to visualize the cross-validated error for different values of your parameter.

In [42]:
```
plot(lasso.cv)
```



(c) Let's use the 1-standard-error rule to pick the tuning parameter λ. Set

```
my.lambda = lasso.cv$lambda.1se
```

Print the value of the `my.lambda`.

In [43]:
```
my.lambda = lasso.cv$lambda.1se
my.lambda
```

0.0712094577138262

(d) Display the predictors with non-zero coefficients via

```
nonzero = predict(lasso.cv, s="lambda.1se", type="nonzero")
colnames(train.X)[nonzero]
```

Out of the 323 original predictors, how many have non-zero coefficients?

In [47]:
```
nonzero = predict(lasso.cv, type="nonzero", s='lambda.1se')
dim(nonzero)

# colnames(train.X)[nonzero]
# error: invalid subscript type 'list'
colnames(train.X)[as.numeric(unlist(nonzero))]

# 9 non-zero coefficients
colnames(train.X)[nonzero[,]]
```

```
9  1
```

'Onset.Delta' 'sd.alsfrs.score' 'alsfrs.score.slope' 'last.speech' 'fvc.liters.slope' 'min.slope.alsfrs.score' 'sum.slope.alsfrs.score' 'min.slope.speech' 'sum.slope.speech'

'Onset.Delta' 'sd.alsfrs.score' 'alsfrs.score.slope' 'last.speech' 'fvc.liters.slope' 'min.slope.alsfrs.score' 'sum.slope.alsfrs.score' 'min.slope.speech' 'sum.slope.speech'

(e) Compute the test `RMSE` for the model fit in part (b), using the regularization parameter chosen in part (c).

Note that it performs significantly better than the linear regression from Homework 1 (which had an `RMSE` of 0.7527). In the ALS Prediction Prize4Life Challenge mentioned in Homework 1, the test error from the Lasso model alone already yielded a very competitive score for the challenge!

In [48]:
```
lasso.test.pred = predict(lasso.cv, as.matrix(test.X), s='lambda.1se')
sqrt(mean((lasso.test.pred - test.y)^2))
```

```
0.526945706527386
```

In [17]:
```
# wrong
tx = model.matrix(test.y~., test.X)
lasso.pred = predict(lasso.mod, s=bestlam, newx=tx)
mean((lasso.pred-test.y)^2)
```

```
0.278140092741145
```

(f) Repeat parts (a,d,e) using ridge regression instead of the Lasso. Set again the seed to the value 2017 before calling `cv.glmnet()`. Store the result of cv.glmnet() in a variable called ridge.cv. Comment on your findings.

In [50]:
```
set.seed(2017)
ridge.cv = cv.glmnet(x=as.matrix(train.X), y=train.y, alpha=0)

nonzero.ridge = predict(ridge.cv, s='lambda.1se', type ='nonzero')
colnames(train.X)[nonzero.ridge[,]]

ridge.test.pred = predict(ridge.cv, as.matrix(test.X), s='lambda.1se')
sqrt(mean((ridge.test.pred - test.y)^2))
```

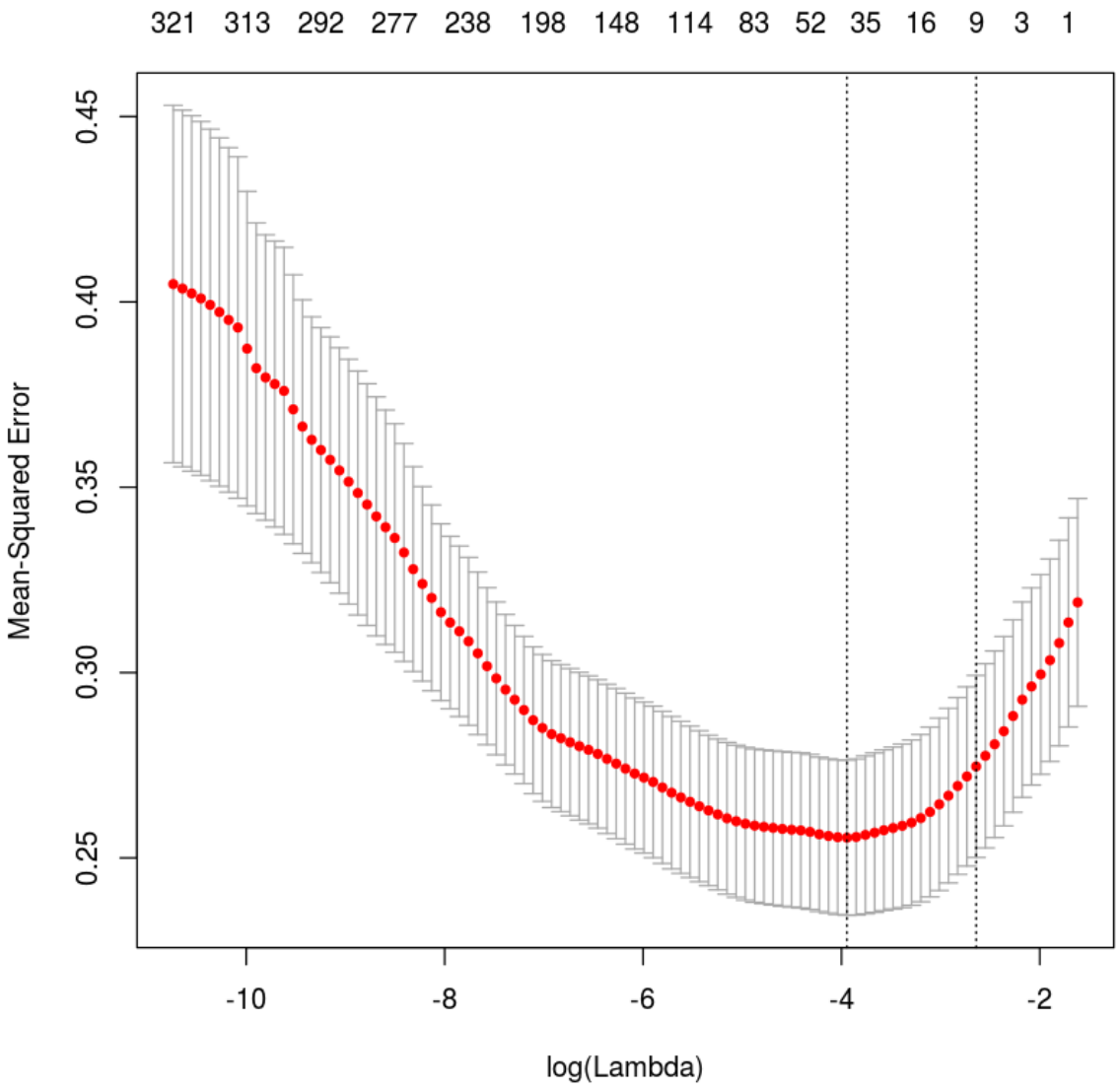
'Onset.Delta' 'Symptom.Speech' 'Symptom.WEAKNESS'
'Site.of.Onset.Onset..Bulbar' 'Site.of.Onset.Onset..Limb' 'Race...Caucasian'
'Age' 'Sex.Female' 'Sex.Male' 'Mother' 'Family' 'Study.Arm.PLACEBO'
'Study.Arm.ACTIVE' 'max.alsfrs.score' 'min.alsfrs.score' 'last.alsfrs.score'
'mean.alsfrs.score' 'num.alsfrs.score.visits' 'sum.alsfrs.score'
'first.alsfrs.score.date' 'last.alsfrs.score.date' 'meansquares.alsfrs.score'
'sd.alsfrs.score' 'alsfrs.score.slope' 'max.speech' 'min.speech' 'last.speech'
'mean.speech' 'sum.speech' 'meansquares.speech' 'sd.speech'
'speech.slope' 'max.salivation' 'min.salivation' 'last.salivation' 'mean.salivation'
'sum.salivation' 'meansquares.salivation' 'sd.salivation' 'salivation.slope'
'max.swallowing' 'min.swallowing' 'last.swallowing' 'mean.swallowing'
'sum.swallowing' 'meansquares.swallowing' 'sd.swallowing' 'swallowing.slope'
'max.handwriting' 'min.handwriting' 'last.handwriting' 'mean.handwriting'
'sum.handwriting' 'meansquares.handwriting' 'sd.handwriting'
'handwriting.slope' 'max.cutting' 'min.cutting' 'last.cutting' 'mean.cutting'
'sum.cutting' 'meansquares.cutting' 'sd.cutting' 'cutting.slope' 'max.dressing'
'min.dressing' 'last.dressing' 'mean.dressing' 'sum.dressing'
'meansquares.dressing' 'sd.dressing' 'dressing.slope' 'max.turning'
'min.turning' 'last.turning' 'mean.turning' 'sum.turning' 'meansquares.turning'
'sd.turning' 'turning.slope' 'max.walking' 'min.walking' 'last.walking'
'mean.walking' 'sum.walking' 'meansquares.walking' 'sd.walking'
'walking.slope' 'max.climbing.stairs' 'min.climbing.stairs' 'last.climbing.stairs'
'mean.climbing.stairs' 'sum.climbing.stairs' 'meansquares.climbing.stairs'
'sd.climbing.stairs' 'climbing.stairs.slope' 'max.fvc.liters' 'min.fvc.liters'
'last.fvc.liters' 'mean.fvc.liters' 'num.fvc.liters.visits' 'sum.fvc.liters'
'last.fvc.liters.date' 'meansquares.fvc.liters' 'sd.fvc.liters' 'fvc.liters.slope'
'lessthan2.fvc.liters' 'no.fvc.liters.data' 'max.svc.liters' 'min.svc.liters'
'last.svc.liters' 'mean.svc.liters' 'num.svc.liters.visits' 'sum.svc.liters'
'last.svc.liters.date' 'meansquares.svc.liters' 'sd.svc.liters' 'svc.liters.slope'
'max.weight' 'min.weight' 'last.weight' 'mean.weight' 'num.weight.visits'
'sum.weight' 'last.weight.date' 'meansquares.weight' 'sd.weight' 'weight.slope'
'lessthan2.weight' 'max.height' 'min.height' 'mean.height' 'num.height.visits'
'sum.height' 'last.height.date' 'meansquares.height' 'no.height.data'
'max.resp.rate' 'min.resp.rate' 'last.resp.rate' 'mean.resp.rate'
'num.resp.rate.visits' 'sum.resp.rate' 'last.resp.rate.date'
'meansquares.resp.rate' 'sd.resp.rate' 'resp.rate.slope' 'lessthan2.resp.rate'
'no.resp.rate.data' 'max.bp.diastolic' 'min.bp.diastolic' 'last.bp.diastolic'
'mean.bp.diastolic' 'num.bp.diastolic.visits' 'sum.bp.diastolic'
'last.bp.diastolic.date' 'meansquares.bp.diastolic' 'sd.bp.diastolic'
'bp.diastolic.slope' 'lessthan2.bp.diastolic' 'no.bp.diastolic.data'
'max.bp.systolic' 'min.bp.systolic' 'last.bp.systolic' 'mean.bp.systolic'
'num.bp.systolic.visits' 'sum.bp.systolic' 'meansquares.bp.systolic'
'sd.bp.systolic' 'bp.systolic.slope' 'max.slope.alsfrs.score'
'min.slope.alsfrs.score' 'last.slope.alsfrs.score' 'mean.slope.alsfrs.score'
'num.slope.alsfrs.score.visits' 'sum.slope.alsfrs.score'
'first.slope.alsfrs.score.date' 'last.slope.alsfrs.score.date'
'meansquares.slope.alsfrs.score' 'sd.slope.alsfrs.score' 'slope.alsfrs.score.slope'
'max.slope.speech' 'min.slope.speech' 'last.slope.speech' 'mean.slope.speech'

'sum.slope.speech' 'meansquares.slope.speech' 'sd.slope.speech'
'slope.speech.slope' 'max.slope.salivation' 'min.slope.salivation'
'last.slope.salivation' 'mean.slope.salivation' 'sum.slope.salivation'
'meansquares.slope.salivation' 'sd.slope.salivation' 'slope.salivation.slope'
'max.slope.swallowing' 'min.slope.swallowing' 'last.slope.swallowing'
'mean.slope.swallowing' 'sum.slope.swallowing' 'meansquares.slope.swallowing'
'sd.slope.swallowing' 'slope.swallowing.slope' 'max.slope.handwriting'
'min.slope.handwriting' 'last.slope.handwriting' 'mean.slope.handwriting'
'sum.slope.handwriting' 'meansquares.slope.handwriting' 'sd.slope.handwriting'
'slope.handwriting.slope' 'max.slope.cutting' 'min.slope.cutting'
'last.slope.cutting' 'mean.slope.cutting' 'sum.slope.cutting'
'meansquares.slope.cutting' 'sd.slope.cutting' 'slope.cutting.slope'
'max.slope.dressing' 'min.slope.dressing' 'last.slope.dressing'
'mean.slope.dressing' 'sum.slope.dressing' 'meansquares.slope.dressing'
'sd.slope.dressing' 'slope.dressing.slope' 'max.slope.turning' 'min.slope.turning'
'last.slope.turning' 'mean.slope.turning' 'sum.slope.turning'
'meansquares.slope.turning' 'sd.slope.turning' 'slope.turning.slope'
'max.slope.walking' 'min.slope.walking' 'last.slope.walking'
'mean.slope.walking' 'sum.slope.walking' 'meansquares.slope.walking'
'sd.slope.walking' 'slope.walking.slope' 'max.slope.climbing.stairs'
'min.slope.climbing.stairs' 'last.slope.climbing.stairs' 'mean.slope.climbing.stairs'
'sum.slope.climbing.stairs' 'meansquares.slope.climbing.stairs'
'sd.slope.climbing.stairs' 'slope.climbing.stairs.slope' 'max.slope.fvc.liters'
'min.slope.fvc.liters' 'last.slope.fvc.liters' 'mean.slope.fvc.liters'
'num.slope.fvc.liters.visits' 'sum.slope.fvc.liters' 'first.slope.fvc.liters.date'
'last.slope.fvc.liters.date' 'meansquares.slope.fvc.liters' 'sd.slope.fvc.liters'
'slope.fvc.liters.slope' 'lessthan2.slope.fvc.liters' 'no.slope.fvc.liters.data'
'max.slope.svc.liters' 'min.slope.svc.liters' 'last.slope.svc.liters'
'mean.slope.svc.liters' 'first.slope.svc.liters.date' 'last.slope.svc.liters.date'
'meansquares.slope.svc.liters' 'sd.slope.svc.liters' 'slope.svc.liters.slope'
'max.slope.weight' 'min.slope.weight' 'last.slope.weight' 'mean.slope.weight'
'num.slope.weight.visits' 'sum.slope.weight' 'first.slope.weight.date'
'last.slope.weight.date' 'meansquares.slope.weight' 'sd.slope.weight'
'slope.weight.slope' 'lessthan2.slope.weight' 'first.slope.height.date'
'max.slope.resp.rate' 'min.slope.resp.rate' 'last.slope.resp.rate'
'mean.slope.resp.rate' 'num.slope.resp.rate.visits' 'sum.slope.resp.rate'
'first.slope.resp.rate.date' 'last.slope.resp.rate.date' 'meansquares.slope.resp.rate'
'sd.slope.resp.rate' 'slope.resp.rate.slope' 'lessthan2.slope.resp.rate'
'no.slope.resp.rate.data' 'max.slope.bp.diastolic' 'min.slope.bp.diastolic'
'last.slope.bp.diastolic' 'mean.slope.bp.diastolic' 'num.slope.bp.diastolic.visits'
'sum.slope.bp.diastolic' 'first.slope.bp.diastolic.date' 'last.slope.bp.diastolic.date'
'meansquares.slope.bp.diastolic' 'sd.slope.bp.diastolic' 'slope.bp.diastolic.slope'
'lessthan2.slope.bp.diastolic' 'max.slope.bp.systolic' 'min.slope.bp.systolic'
'last.slope.bp.systolic' 'mean.slope.bp.systolic' 'num.slope.bp.systolic.visits'
'sum.slope.bp.systolic' 'first.slope.bp.systolic.date'
'meansquares.slope.bp.systolic' 'sd.slope.bp.systolic' 'slope.bp.systolic.slope'

0.541172112545499

In [49]:
```
# wrong

# x = model.matrix(train.y~., train.X)
# y = train.y

# ridge.mod = glmnet(x, y, alpha=0)

# set.seed(2017)
# ridge.cv = cv.glmnet(x, y, alpha=1)

# bestlam.ridge = ridge.cv$lambda.min
# bestlam.ridge

# ridge.coef = predict(ridge.mod, type="coefficients", s=bestlam.ridge)
# head(ridge.coef)

# my.lambda.ridge = ridge.cv$lambda.1se
# my.lambda.ridge
```

In [255]:
```
plot(ridge.cv)
```



Gyu-Ho's Answer: Above models in Ridge Regression and Lasso return **same best lambda value**. **Same number of non-zero coefficients**, *9*, according to `predict(type="nonzero")` results. However, ridge regression coefficients in training data are only shrunken towards zero, while all variables are still included in the model. Lasso performs variable selection, with coefficients being estimated exactly to zero, thus easier to interpret. On this test data, Lass has less `RMSE`, thus performs better.

All 323 predictors are nonzero. This is not surprising, as ridge regression does not induce sparse models. The RMSE is 0.541, which is very similar to the lasso RMSE.