MMSS 311-2 HW1

Yushi Liu

Regression

OLS

```
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
```

```
library(ggplot2)
sick <- read.csv("sick_data.csv")
sick$dummy <- ifelse(sick$result == "Positive", 1, 0)
lm1 <- lm(dummy~bp+temp, data=sick)
summary(lm1)
```

```
##
## Call:
## lm(formula = dummy ~ bp + temp, data = sick)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.32785 -0.09918 -0.02229  0.05700  0.82096
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.2134563  0.5141439  -10.14   <2e-16 ***
## bp          -0.0082865  0.0004702  -17.62   <2e-16 ***
## temp         0.0628185  0.0050579   12.42   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1695 on 997 degrees of freedom
## Multiple R-squared:  0.3966, Adjusted R-squared:  0.3954
## F-statistic: 327.7 on 2 and 997 DF,  p-value: < 2.2e-16
```

```r
sick$predicted <- ifelse (fitted(lm1) < 0.5, "Negative", "Positive")
sick$count <- ifelse (sick$result == sick$predicted, 1, 0)
sum(sick$count)/1000
```

```
## [1] 0.964
```

Logit

```r
lm2 <- glm(dummy~bp+temp, data=sick, family = "binomial")
summary(lm2)
```

```
##
## Call:
## glm(formula = dummy ~ bp + temp, family = "binomial", data = sick)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.62332  -0.02253  -0.00462  -0.00093   3.02311
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -199.3267    46.8077  -4.258 2.06e-05 ***
## bp            -0.3499     0.0638  -5.485 4.14e-08 ***
## temp           2.3140     0.4923   4.700 2.60e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 397.030  on 999  degrees of freedom
## Residual deviance:  53.837  on 997  degrees of freedom
## AIC: 59.837
##
## Number of Fisher Scoring iterations: 10
```

```r
sick$predicted_logit <- ifelse (fitted(lm2) < 0.5, "Negative", "Positive")
sick$count_logit <- ifelse (sick$result == sick$predicted_logit, 1, 0)
sum(sick$count_logit)/1000
```

```
## [1] 0.992
```

```r
c <- (0.5 - coef(lm2)["(Intercept)"])/coef(lm2)["temp"]
d <- coef(lm2)["bp"]/coef(lm2)["temp"]
```

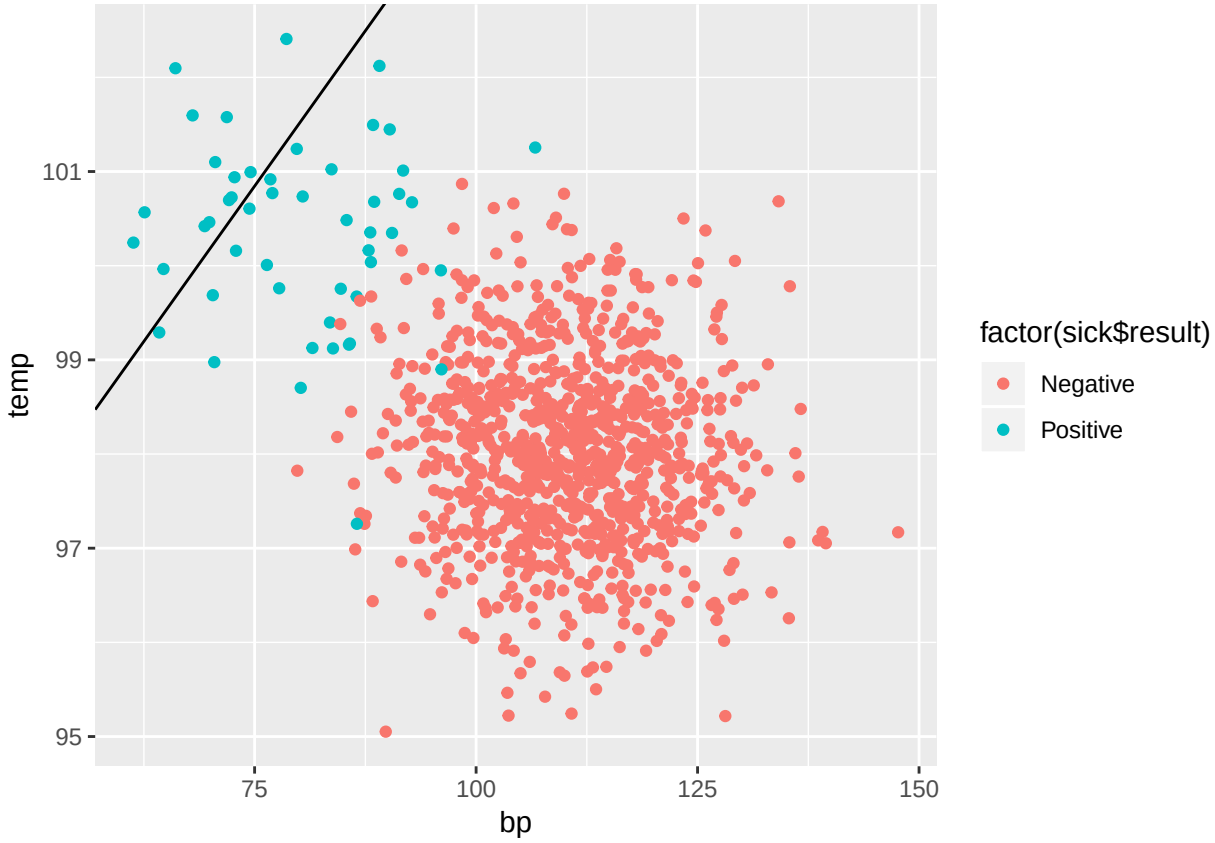
b) The logit regression predicts 99.2% of the cases correctly, while the linear regression predicts 96.4% correctly. Statistically, the logit model predicts more accurately than the the linear regression.

c) For the OLS model: $bp = 90.95175 + 0.1319124temp$.

For the logit model: $bp = -571.009822 + 6.612235temp$

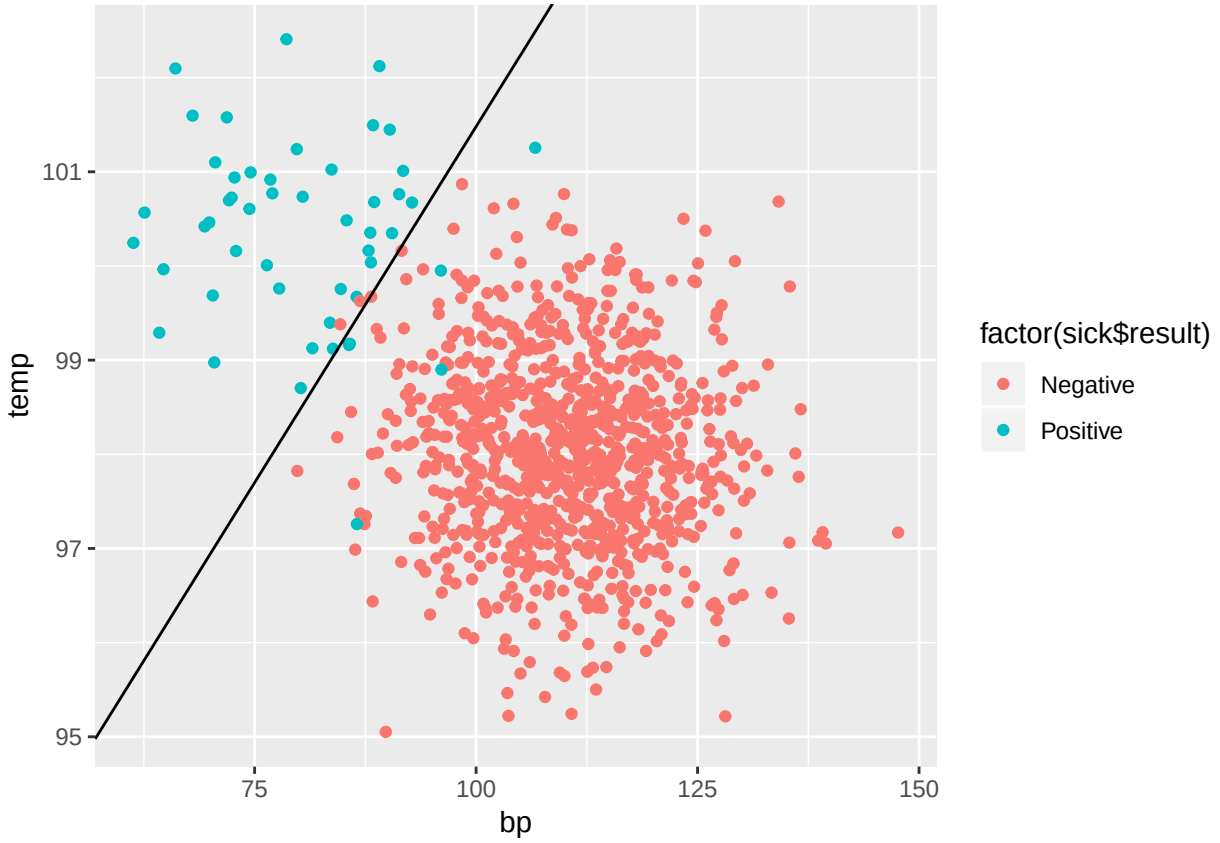
d) Below is the graph with OLS line:

```
a <- (0.5 - coef(lm1)["(Intercept)"])/coef(lm1)["temp"]
b <- coef(lm1)["bp"]/coef(lm1)["temp"]
ggplot(sick, aes(x = bp, y= temp))+
  geom_point(aes(color = factor(sick$result ))) +
  geom_abline(intercept = a, slope = -b, color = "black")
```



Below is the graph with logit line:

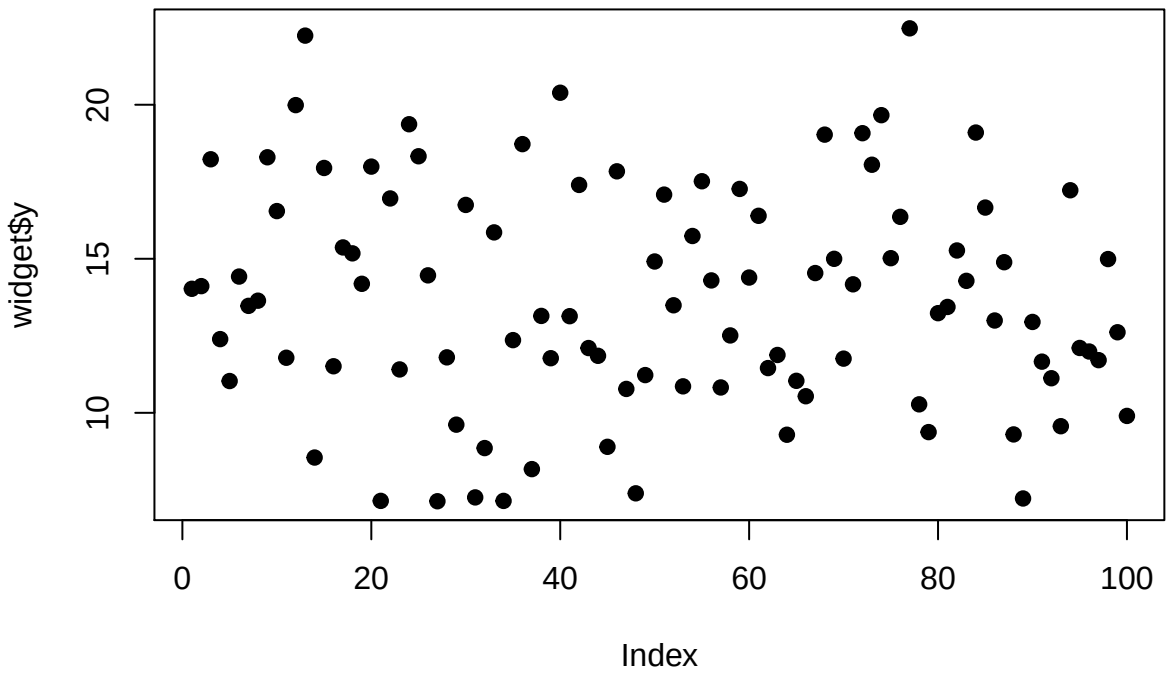
```
ggplot(sick, aes(x = bp, y= temp))+
  geom_point(aes(color = factor(sick$result ))) +
  geom_abline(intercept = c, slope = -d, color = "black")
```



Regularization/Selection

a)

```
widget <- read.csv("widget_data.csv")
plot(widget$y, pch=19)
```



Ridge Regression b)

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-16
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.2.1 --
```

```
## v tibble  2.0.1     v purrr   0.2.5
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

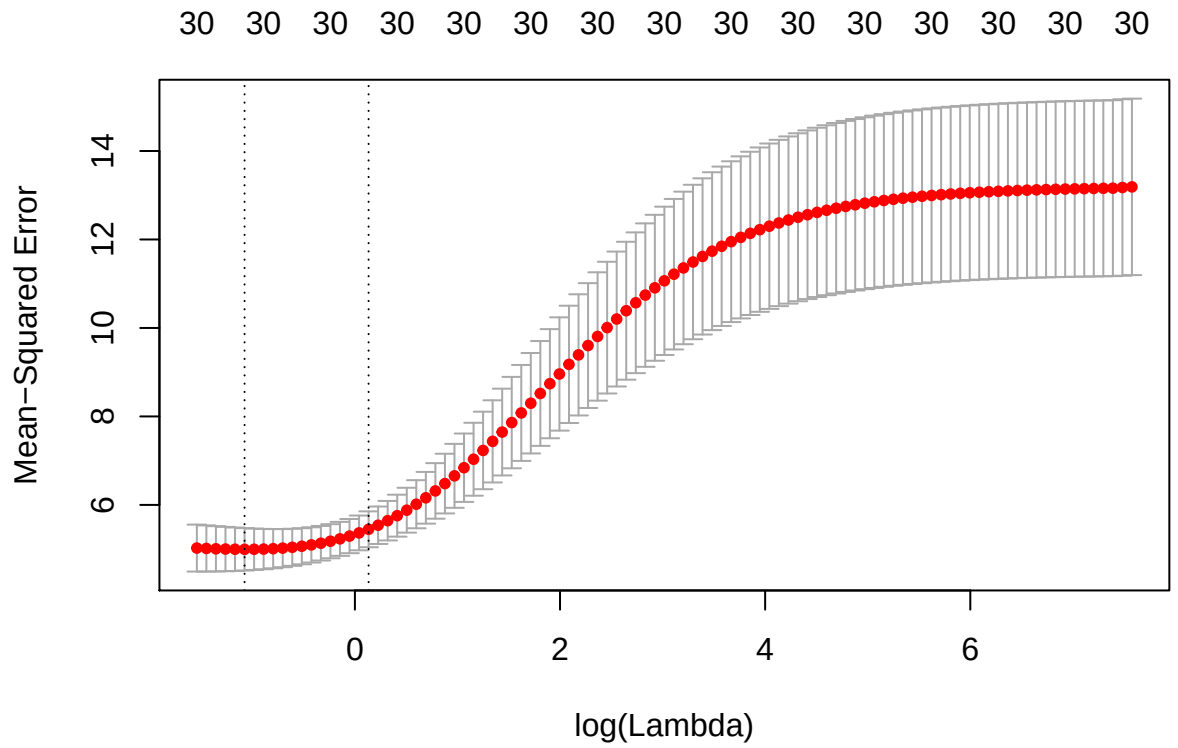
```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x purrr::accumulate() masks foreach::accumulate()
## x tidyr::expand()     masks Matrix::expand()
## x dplyr::filter()     masks stats::filter()
## x dplyr::lag()        masks stats::lag()
## x purrr::when()       masks foreach::when()
```

```
library(broom)

grid = 10^seq(2, -2, length = 100)
ridge.mod = glmnet(x = as.matrix(widget[,2:31]), y = as.matrix(widget[1]), alpha = 0, lambda=grid) %>% t

#Plot Ridge
ridge.plot = ggplot(data.frame(ridge.mod), aes(x = ridge.mod$lambda, y = ridge.mod$estimate)) + geom_poi

#CV Ridge
set.seed(1)
cv.out = cv.glmnet(x = as.matrix(widget[,2:31]), y = as.matrix(widget[1]), alpha = 0)
plot(cv.out)
```




```
bestlam = cv.out$lambda.min
bestlam
```

```
## [1] 0.3410022
```

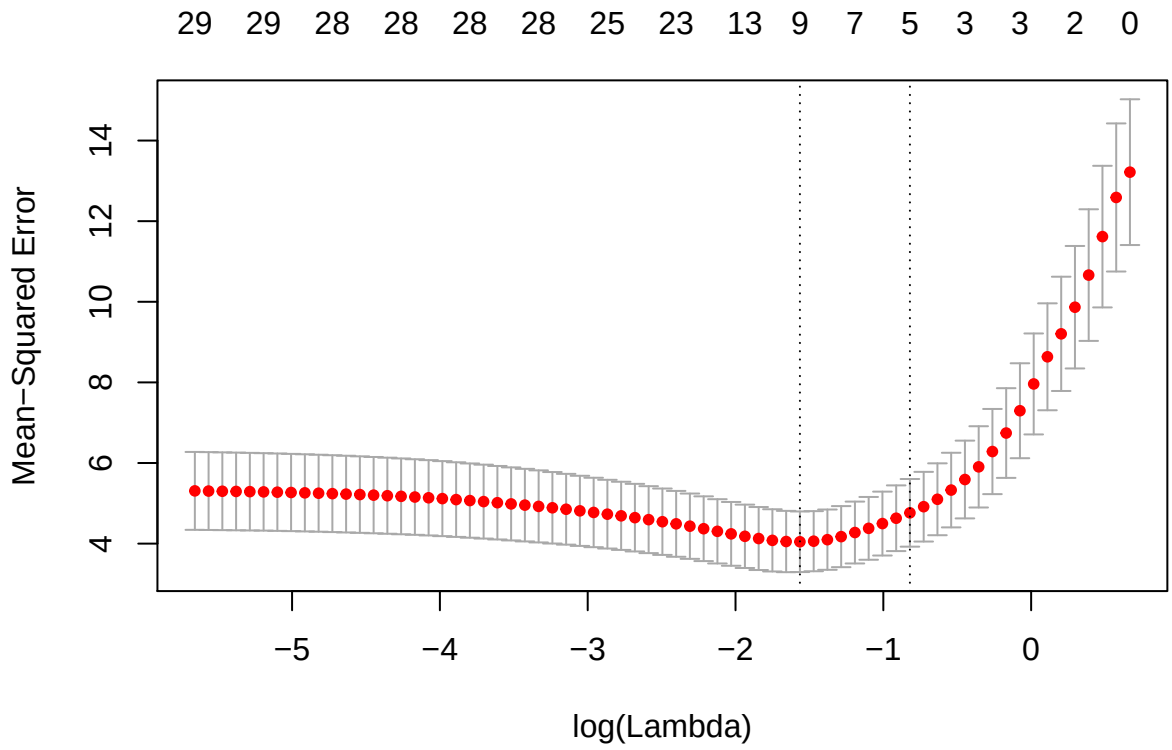
The best lambda is 0.3410022.

Lasso

```
lasso.mod = glmnet(x = as.matrix(widget[, 2:31]), y = as.matrix(widget[1]), alpha = 1, lambda=grid) %>%

#Plot lasso
lasso.plot = ggplot(data.frame(lasso.mod), aes(x = lasso.mod$lambda, y = lasso.mod$estimate)) + geom_po

#CV lasso
set.seed(2)
cv.out1 = cv.glmnet(x = as.matrix(widget[,2:31]), y = as.matrix(widget[1]), alpha = 1)
plot(cv.out1)
```




```
bestlam1 = cv.out1$lambda.min
bestlam1
```

```
## [1] 0.2092358
```

The best lambda for the lasso regression is 0.2092358. The lasso regression gives a smaller lambda than ridge regression does. Lasso regression and ridge regression give different optimal lambdas because they impose different rules of deciding penalties. # Classification

```
library(e1071)
pol_data = read.csv("pol_data.csv")
# Split the data
pol_data["ID"] <- rownames(pol_data)
pol_data$dummy <- ifelse(pol_data$group == "Socialcrat", 1, 0)
```

```
set.seed(20)
test <- pol_data[sample(nrow(pol_data), 100), ]
train <- pol_data[!(pol_data$ID %in% test$ID),]
```

Naive Bayes

```
dat <- data.frame(x = train[,2:4], y = as.factor(train$dummy))
nb <- naiveBayes(y ~ ., data = dat, na.action = na.pass)
testdat <- data.frame(x=test[2:4], y = as.factor(test$dummy))
NB.prediction <- predict(nb, testdat)
table(predict = NB.prediction, truth = testdat$y)
```

```
##         truth
## predict  0  1
##       0 44  2
##       1  2 52
```

SVM

```
svmfit <- svm(y~., data=dat, kernel="linear", cost=10,
scale=FALSE)
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.3333333
##
## Number of Support Vectors:  22
##
##  ( 11 11 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
#SVM tune out
set.seed(250)
tune.out = tune(svm, y~., data = dat, kernel = "linear",ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10,100
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.03
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03 0.520 0.14567849
## 2 1e-02 0.045 0.06851602
## 3 1e-01 0.030 0.06324555
## 4 1e+00 0.030 0.06324555
## 5 5e+00 0.030 0.06324555
## 6 1e+01 0.030 0.06324555
## 7 1e+02 0.030 0.06324555
```

```
bestmod = tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##       gamma:  0.3333333
##
## Number of Support Vectors:  44
##
##  ( 22 22 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
#make prediction using the best model
#testdat = data.frame(x = test[2:4], y = as.factor(test$dummy))
bsvm = svm(train$group~train$pol_margin+train$col_degree+train$house_income, data = train, kernel = "li
ypred <- predict(bestmod, testdat)

table(predict = ypred, truth = testdat$y)
```

```
##        truth
```

```
## predict  0  1
##       0 43  2
##       1  3 52
```