

Oving6

2022-04-04

Task 2

```
library(ISLR)

set.seed(1)

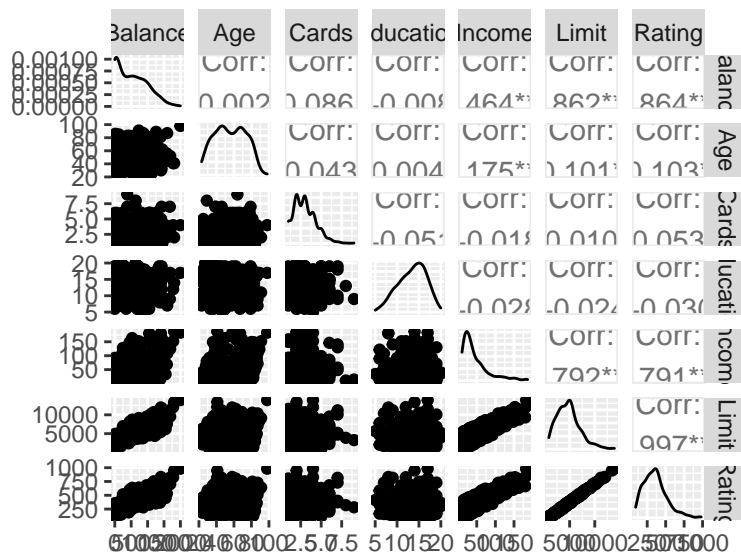
data <- Credit

w_d <- c("Balance", "Age", "Cards", "Education", "Income", "Limit", "Rating")

scatter_data <- Credit[,w_d]

library(GGally)

ggpairs(data=scatter_data)
```



Task 3a

```
#We first want to get rid of the id column

credit_data = subset(Credit, select=-c(ID))

library(leaps)
```

```

variables = dim(credit_data)[2]

index = sample(1:nrow(credit_data), nrow(credit_data)*0.75)

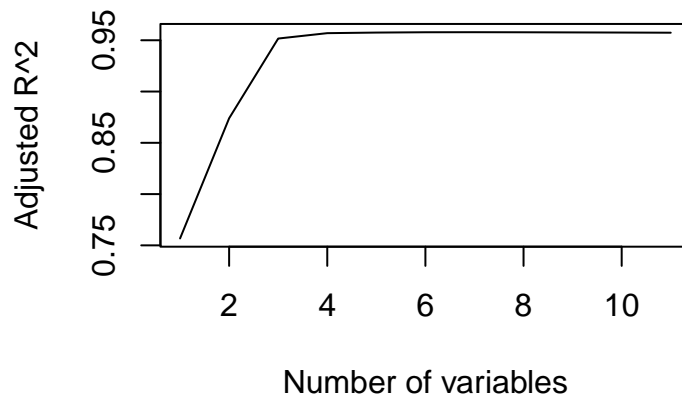
training_data <- credit_data[index,]
testing_data <- credit_data[-index,]

model = regsubsets(Balance~., data = training_data, nvmax = variables)

model_summary = summary(model)

plot(model_summary$adjr2, xlab = "Number of variables", ylab = "Adjusted R^2", type = "l")

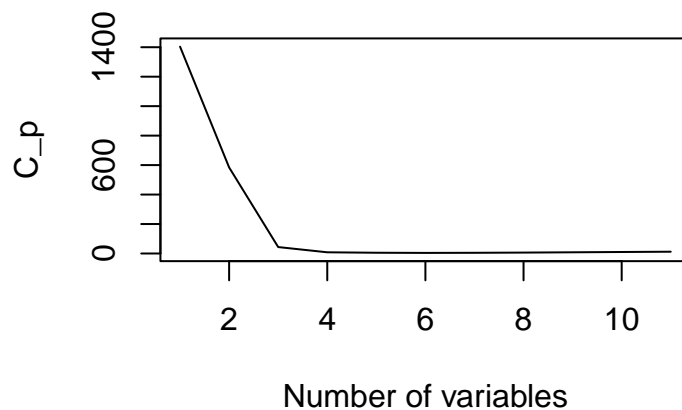
```



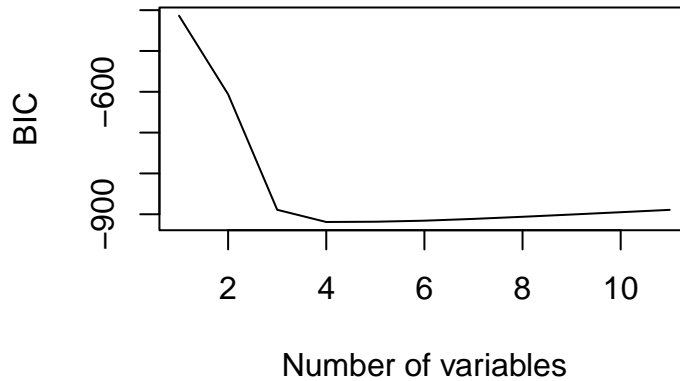
```

plot(model_summary$cp, xlab = "Number of variables", ylab = "C_p", type = "l")

```



```
plot(model_summary$bic, xlab = "Number of variables", ylab = "BIC", type = "l")
```



We observe from the plots that the BIC has a minimum in 4 variables, Cp has a minimum of around 4 variables, and adjusted R^2 has a maximum around 3 variables.

We choose the simplest model with the lowest Cp and BIC, and highest adjusted R^2 , therefore 4 variable model is optimal.

Task 3b

```
K = 10      #We want k = 10 folds

set.seed(1)

#Creating K folds from 300 rows of data

folds = sample(1:K, nrow(training_data), replace=TRUE)

#Making a predict function for the regsubsets

predict.regsubsets=function(object,data, id){
  form = as.formula(object$call[[2]])    #?
  mat = model.matrix(form,data)          #Creates a design matrix
  coefi = coef(object,id=id)             #extracts the coefficients of the models
  xvars = names(coefi)
  mat[,xvars] %*% coefi                  #Multiplies the matrix by the coefficients
}

cv.errors = matrix(NA, K, variables, dimnames = list(NULL, paste(1:variables)))

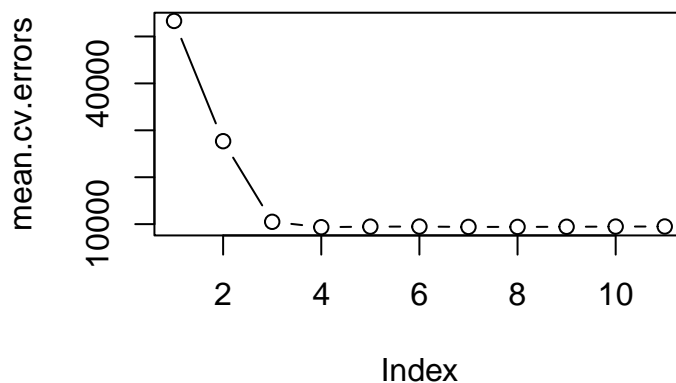
# For each fold
for(j in 1:K){
  # Fit the model with each subset of predictors on the training part of the fold
  best.fit=regsubsets(Balance~.,data=training_data[folds!=j,], nvmax=variables)
  # For each subset
```

```

for(i in 1:variables){
  # Predict on the hold out part of the fold for that subset
  pred=predict(best.fit, training_data[folds==j,],id=i)
  # Get the mean squared error for the model trained on the fold with the subset
  cv.errors[j,i]=mean((training_data$Balance[folds==j]-pred)^2)
}
}
mean.cv.errors = apply(cv.errors,2,mean)

plot(mean.cv.errors,type= "b")

```



We observe that the CV error also reaches a minimum for 4 variables, comparing to the other methods of calculating errors, this seems quite reasonable.

```

#Implementing the optimal model

predictors = 4  #We found this to be the optimal amounts of predictors

variables_chosen = names(coef(best.fit$call[[2]], predictors))  #?
variables_chosen = variables[!variables %in% "(intercept)"]  #?

mformula = as.formula(best.fit$call[[2]])
m_design_matrix = model.matrix(mformula, training_data)[,variables]

m_data_train = data.frame(Balance = training_data$Balance, m_design_matrix)

#Fitting the best model using only the selected predictors on the training data
best_model = lm(formula = mformula, m_data_train)

#Make predictions on test set
m_design_matrix_test = model.matrix(mformula, testing_data)[,variables_chosen]
predictions = predict(object = best_model, data = as.data.frame(m_design_matrix_test))

m_squared_errors = (testing_data$Balance - predictions)^2

```

```
mean(m_squared_errors)
```

```
## [1] 202704.7
```

Task 4

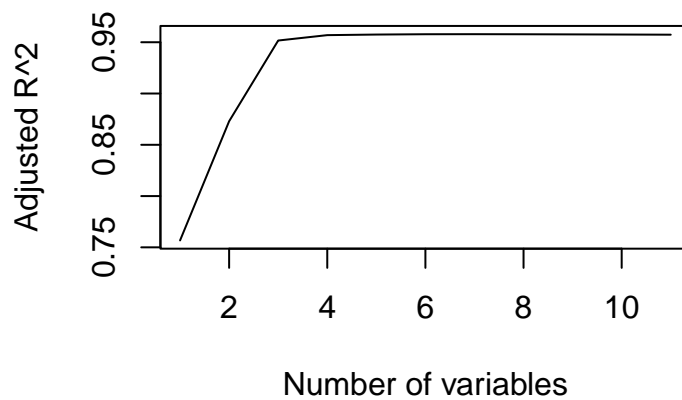
```
#Defining the backward stepwise selection
```

```
backward_model <- regsubsets(Balance~.,data=training_data, nvmax = variables, method = "backward")
```

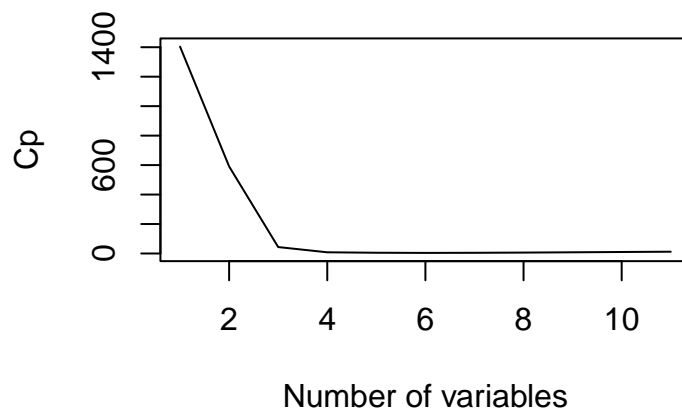
```
#Storing the summary
```

```
summary_backward_model = summary(backward_model)
```

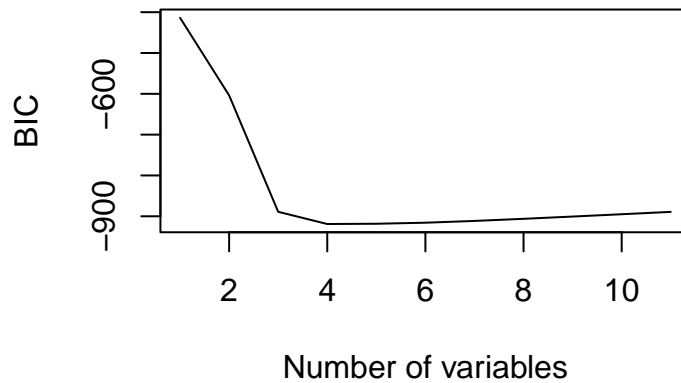
```
plot(summary_backward_model$adjr2, xlab = "Number of variables", ylab = "Adjusted R^2", type= "l")
```



```
plot(summary_backward_model$cp, xlab = "Number of variables", ylab = "Cp", type= "l")
```



```
plot(summary_backward_model$bic, xlab = "Number of variables", ylab = "BIC", type= "l")
```



We see the same results here.

Task 5

Ridge regression is an extension of linear regression where the loss function is modified to minimize the complexity of the model.

$\alpha = 0$ for ridge regression, family = gaussian

```
library(glmnet)

x_train = model.matrix(Balance ~., training_data)[,-1]
y_train = training_data$Balance

x_test = model.matrix(Balance ~., testing_data)[,-1]
y_test = testing_data$Balance

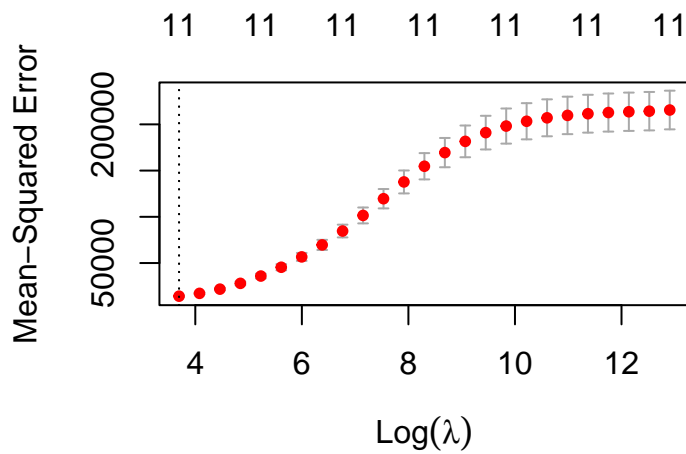
lambdas = 10^seq(2,-3, by = -.1)

ridge_regression = glmnet(x_train,y_train, nlambda = 25, alpha = 0, family = "gaussian")

set.seed(1)

cv = cv.glmnet(x_train,y_train, nlambda = 25, alpha = 0, family = "gaussian")

plot(cv)
```



```
best_lambda = cv$lambda.min

ridge_prediction = predict(ridge_regression, s=best_lambda, newx = x_test)

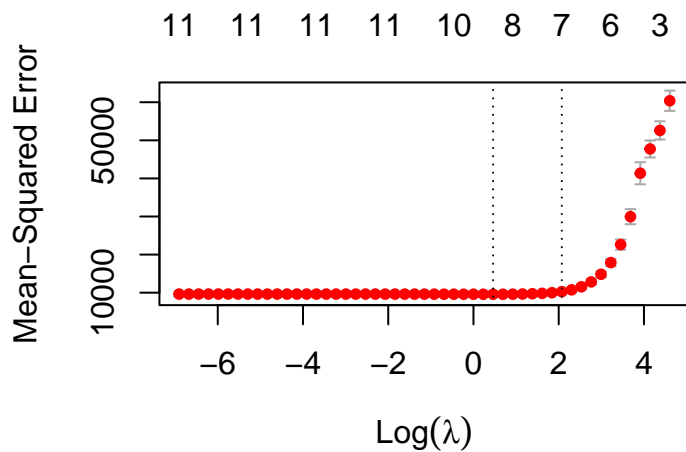
ridge_squared_errors = (y_test - ridge_prediction)^2
mean(ridge_squared_errors)
```

```
## [1] 15741.72
```

Task 6

alpha = 1 for lasso regression

```
lasso_regression = cv.glmnet(x_train, y_train, alpha = 1, lambda = lambdas, standardize = TRUE, nfolds = 10)
plot(lasso_regression)
```



```
lambda_best = lasso_regression$lambda.min

lasso_model = glmnet(x_train, y_train, alpha = 1, lambda = lambda_best, standardize = TRUE)

predictions = predict(lasso_model, s = lambda_best, newx = x_test)

squared_errors = (predictions-y_test)^2
mean(squared_errors)

## [1] 12333.23
```