STAT 218 Analytics Project III

Inigo Benavides and Rommel Bartolome

March 23, 2019

Abstract

We created two models using a dataset collection of 300 sampled water districts in the Philippines. The first model is a regression model with water prices as the output variable while the second one is a categorical model where we created an output variable called wastage rating. The Ridge, Lasso, and Principal Components Regression were employed in creating the models. It has been found that the best model for the regression model is the Lasso Regression Model, with an RMSE of 0.253. On the other hand, the Principal Components Regression Model has been found to be the best model for classification with an AUC of 0.509 and 54% Accuracy.

Introduction

For our third analytics project, we were given the same dataset similar to the first and second project. The data set is comprised of 300 sampled water districts in the Philippines. The specific locations of the water districts have been anonymised and no reference year is provided. There was no autocorrelation, and we will assume that there would be no spatial correlation between districts.

With the same data, we will be creating two models. The first model would be a regression model with the water prices as output variable while the second model would be a categorical model where we created a new output variable called `wastage rating`. For the `wastage rating`, if the percent of non-revenue water from total displaced water (`nrwpercent`) is less than or equal 25, we label it as 1 and 0 otherwise

However, in this project, we will now be employing another set of modelling tools. We will be using Ridge Regression, Lasso Regression and Principal Components Regression. From these three, we will check which has the lowest RMSE and the best accuracy.

Data Loading and Cleaning

Before creating our models, we load all the libraries we will be using in this project:

Similar to our previous project, we will also clean our data and set our seed for reproducibility. Here, we factorize necessary variables and based on previous work, we transform and take the logarithm of `conn` (number of connections in a water district), `vol_nrw` (volume of non-revenue water in cu.m., which is displaced water in which the water district did not collect revenues) and `wd_rate` (water rate in pesos for a specific water district, as minimum charge for the first 10 cu. m.). We also simplify `Mun1` (number of first-class municipalities in the water district) as a binary decision while `conn_p_area` (number of connections per square kilometre) was squared. Lastly, the wastage rating which we will call as `nrwpcent_class` is added for the classfication model.

```
set.seed(1)
df <- read_csv("data_BaBe.csv") %>%
  select(-c(X1)) %>%
  mutate(REGION=as.factor(REGION),
         WD.Area=as.factor(WD.Area),
         Mun1=as.factor(case_when(Mun1 > 0 ~ 1, TRUE ~ 0)),
         conn_log=log(conn),
         vol_nrw_log=log(vol_nrw),
         wd_rate_log=log(wd_rate),
         conn_p_area_squared=conn_p_area^2,
         nrwpcent_class=as.factor(case_when(nrwpcent <= 25 ~ 1, TRUE ~ 0))
```

```r
        # Engineer target classification variable
        )
```

We also created a dummified version of the feature matrix, which we will use later in the classification part. The data was split into a train and a test dataset.

```r
# Create dummified version of feature matrix
df_dummies <- dummyVars(~ ., data=df, fullRank=TRUE) %>% predict(df) %>% as.data.frame()
colnames(df_dummies)[length(df_dummies)] <- "nrwpcent_class"
df_dummies_train <- df_dummies[1:250,]
df_dummies_test <- df_dummies[251:300,]

# Train test split first 250 vs. last 50
df_train <- df[1:250,]
df_test <- df[251:300,]
```

In the following sections, we will explore the fitting of the following models to our water district data set: (1) Ridge Regression, (2) Lasso Regression, and (3) Principal Components Regression. The first part will be for the Regression Model while the latter parts will be for the Categorical Model.

Regression Model

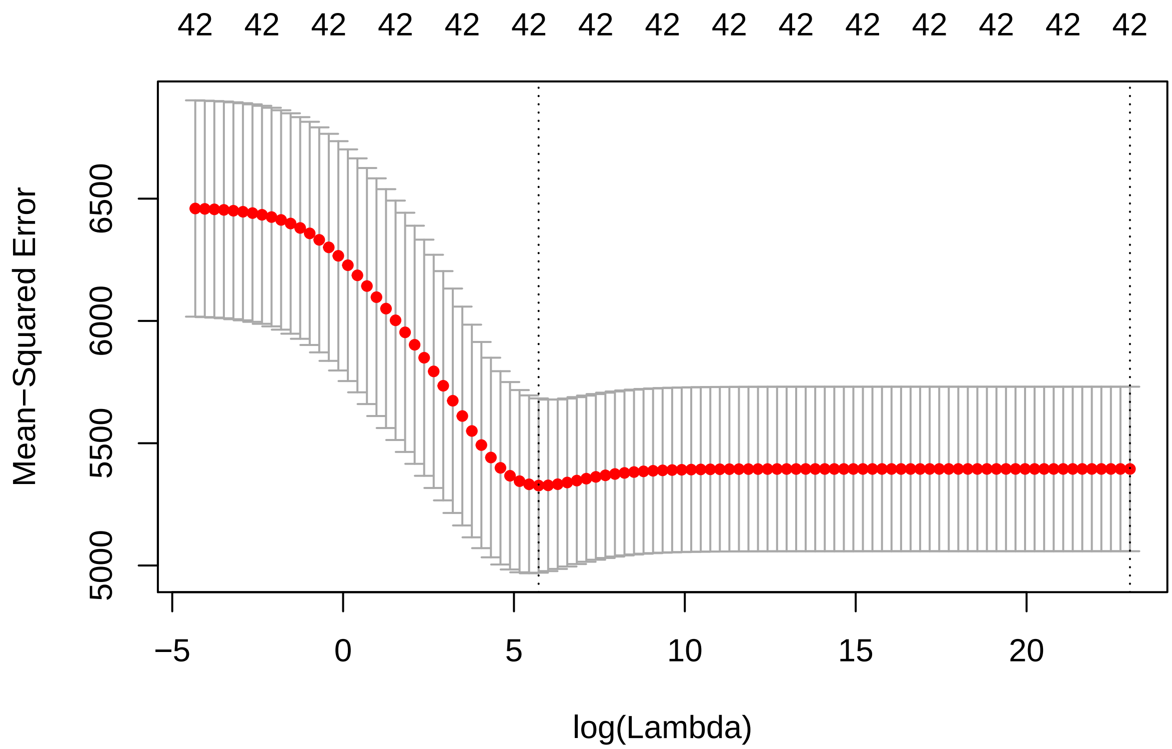
Ridge Regression

For the purpose of this project, we created a helper function called `formulaConstructor` that we will use in coercing our data to the suited form. We also separated the x and the y so we can easily fit it in the regression model later.

```r
formulaConstructor <- function(predictors) {
  predictors %>% paste(collapse=" + ") %>% paste("wd_rate ~", .) %>% as.formula()
}

predictors <- df_train %>% select(-c(wd_rate, wd_rate_log)) %>% names

X_train <- model.matrix(formulaConstructor(predictors), df_train)
y_train <- df_train$wd_rate

X_test <- model.matrix(formulaConstructor(predictors), df_test)
y_test <- df_test$wd_rate
```

Now, we first perform cross validation, so we can choose the value of the tuning parameter lambda.

```r
grid <- 10^seq(10, -2, length=100)
cv_ridge_regression_model <- cv.glmnet(X_train, y_train, alpha=0, lambda=grid)
plot(cv_ridge_regression_model)
```



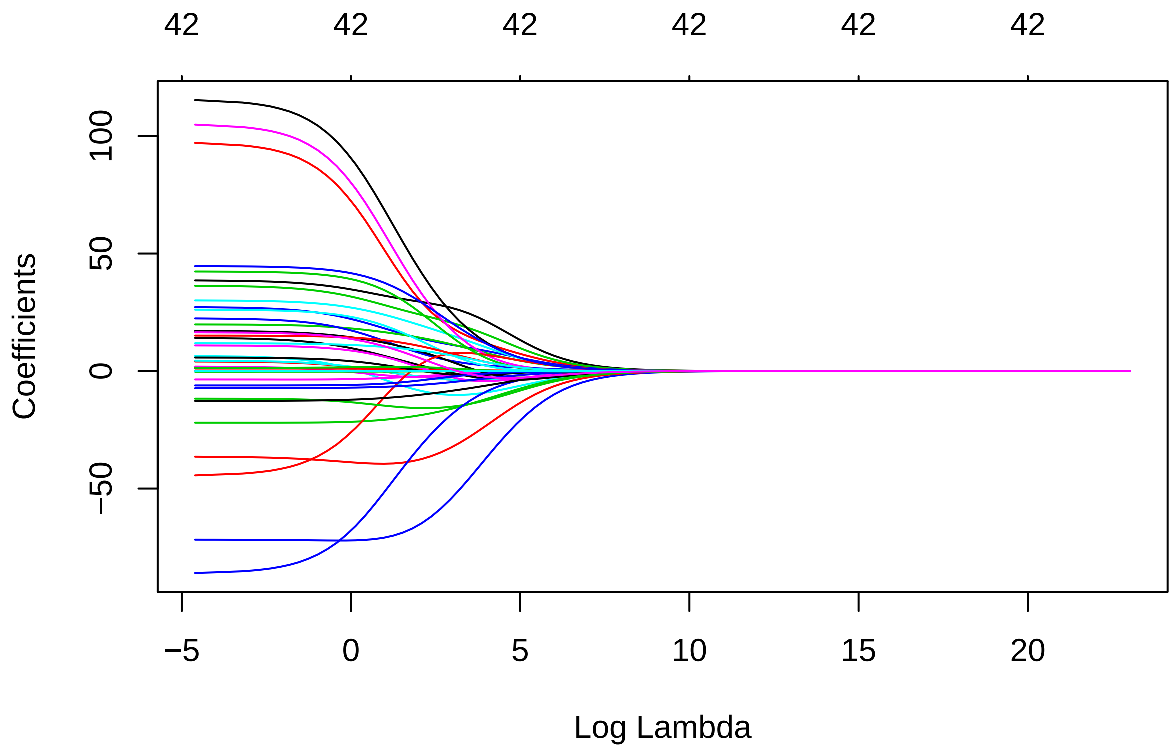
The plot above shows that mean-squared error with respect to the log of lambda.

```
cv_ridge_regression_model$lambda.min
```

```
## [1] 305.3856
```

Based on the above plot, we find that the optimal value of λ that minimizes cross-validation MSE is 305.3856. We also inspecting the model's path coefficients below:

```
plot(cv_ridge_regression_model$glmnet.fit, "lambda", label=FALSE)
```



We now check the performance of the Ridge Regression Model with respect to the train set:

```
# Fit ridge regression model with optimal lambda
optimal_lambda <- cv_ridge_regression_model$lambda.min
ridge_regression_model <- glmnet(X_train, y_train, alpha=0, lambda = optimal_lambda)

# Compute MSE
predictions <- ridge_regression_model %>% predict(X_test) %>% as.vector()
sqrt(mean((predictions - y_test)^2)) / (mean(y_test))
```

```
## [1] 0.2661241
```

We find that our ridge regression model has a test RMSE of 0.2661241.

```
ridge_regression_model_coefs <- ridge_regression_model %>%
  predict(type="coefficients", s=optimal_lambda) %>% as.matrix()
ridge_regression_model_coefs[order(ridge_regression_model_coefs, decreasing=TRUE), ]
```

```
##         (Intercept)           REGIONCAR             REGIONI
##        2.493743e+02        8.082904e+00        6.035359e+00
##        REGIONCARAGA           REGIONII               Mun11
##        4.683880e+00        3.597504e+00        3.102154e+00
##           REGIONIX          REGIONVIII      WD.AreaArea 7
##        3.030311e+00        3.014974e+00        2.865877e+00
##           REGIONVI       WD.AreaArea 3            REGIONX
##        2.732108e+00        1.014583e+00        8.116424e-01
##               Mun5                surw      WD.AreaArea 5
##        6.897713e-01        6.471633e-01        5.150139e-01
```

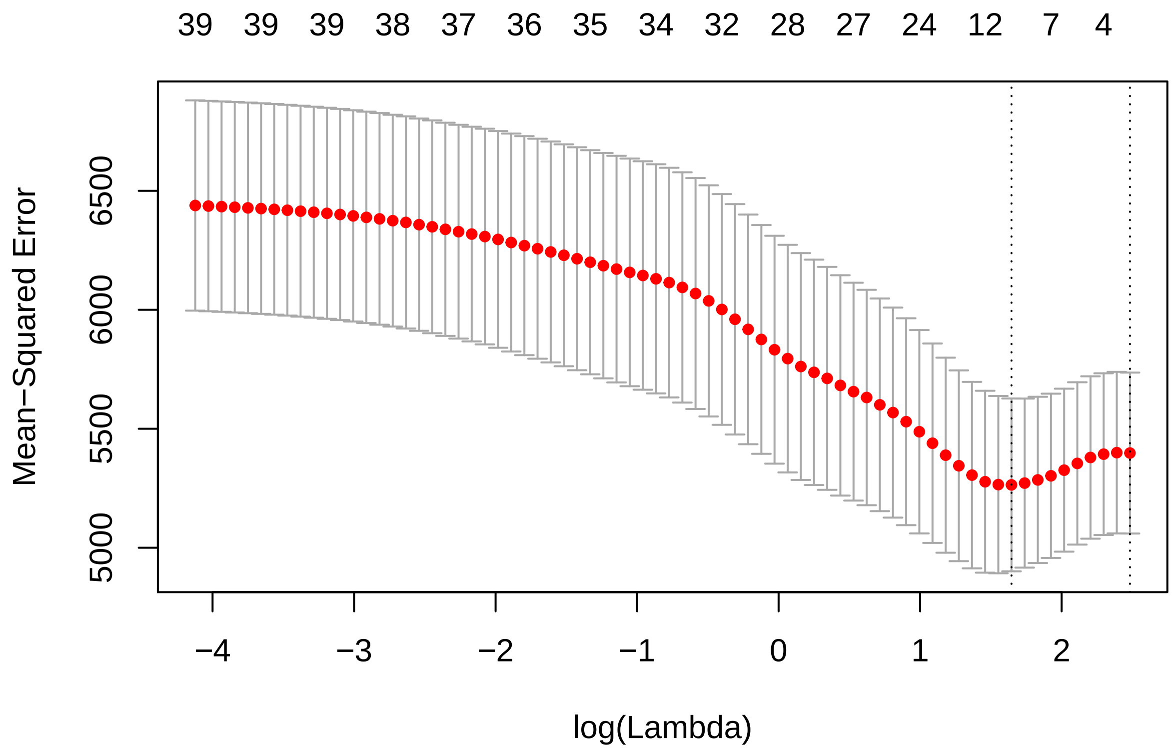
```
##           REGIONIV              cities            nrwpcent
##       2.991376e-01        2.413665e-01        1.140471e-01
##               Mun2         (Intercept) conn_p_area_squared
##       7.885779e-03        0.000000e+00       -1.464665e-08
##            vol_nrw              elevar                conn
##      -4.785423e-07       -1.777770e-05       -5.012102e-05
##        conn_p_area                 emp                  gw
##      -1.797590e-03       -4.013522e-03       -2.877596e-02
##           conn_log         vol_nrw_log                Mun4
##      -2.554640e-01       -2.632432e-01       -3.069151e-01
##               sprw                Mun3      WD.AreaArea 9
##      -9.936188e-01       -1.800178e+00       -1.925625e+00
##            REGIONV       WD.AreaArea 4      nrwpcent_class1
##      -2.146246e+00       -2.146946e+00       -2.171815e+00
##            coastal       WD.AreaArea 6            REGIONXI
##      -2.321429e+00       -2.476884e+00       -3.011723e+00
##          REGIONIII       WD.AreaArea 2       WD.AreaArea 8
##      -4.125936e+00       -4.743344e+00       -5.404018e+00
##          REGIONXII          REGIONVII
##      -8.057085e+00       -1.246343e+01
```

In terms of the resulting coefficients, we find that `REGIONCAR`, `REGIONI`, `REGIONCARAGA` have the largest positive contribution to `wd_rate`, while `REGIONVII`, `REGIONXII` and `WD.AreaArea 8` have the largest negative contribution to `wd_rate`.

Lasso Regression

In this section, we fit a lasso regression model, setting alpha to 1. Again, using cross validation:

```r
set.seed(1)
cv_lasso_regression_model <- cv.glmnet(X_train, y_train, alpha=1)
plot(cv_lasso_regression_model)
```



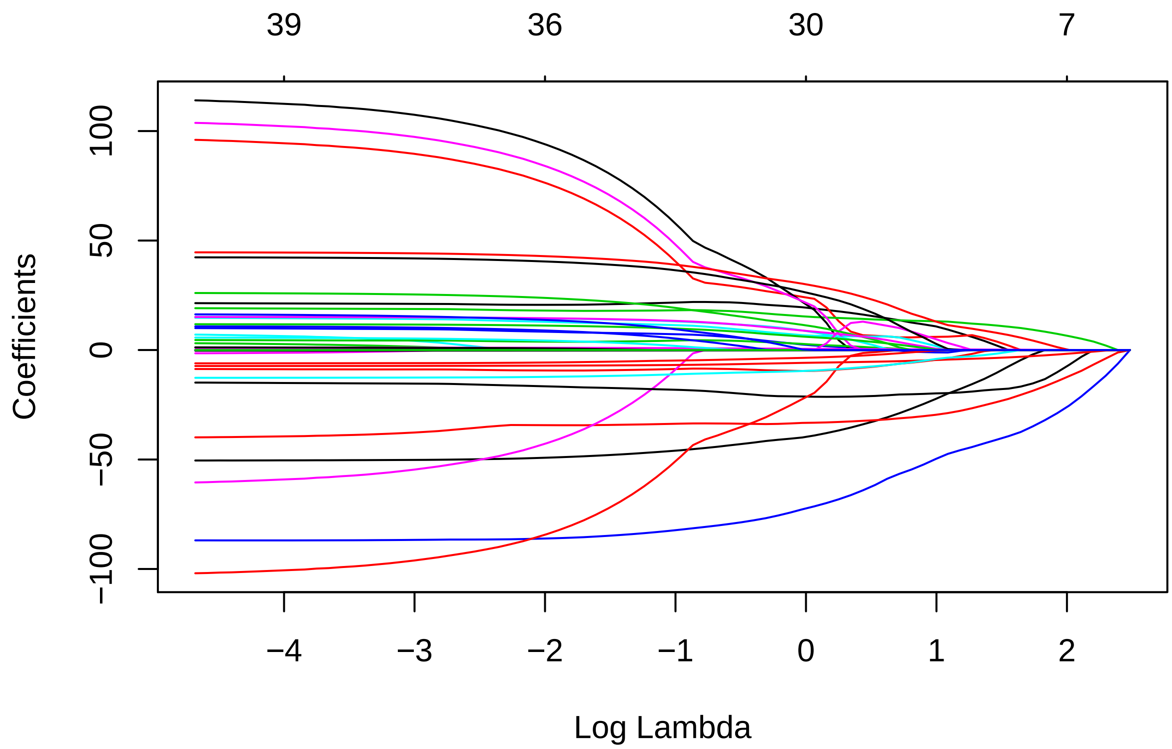
Based on 10-fold cross-validation, we find that a $\lambda = 5.185112$ minimizes CV MSE.

```
cv_lasso_regression_model$lambda.min
```

```
## [1] 5.185112
```

Further inspecting the model's path coefficients:

```
plot(cv_lasso_regression_model$glmnet.fit, "lambda", label=FALSE)
```



We now test the performance of the lasso regression model:

```
# Fit lasso regression model with optimal lambda
optimal_lambda <- cv_lasso_regression_model$lambda.min
lasso_regression_model <- glmnet(X_train, y_train, alpha=0,
                                  lambda = optimal_lambda)

# Compute MSE
predictions <- lasso_regression_model %>%
  predict(X_test) %>% as.vector()
sqrt(mean((predictions - y_test)^2)) / (mean(y_test))
```

```
## [1] 0.2568057
```

We find that our lasso regression model has a test RMSE of 0.2531856, outperforming the ridge regression model.

```
lasso_regression_model_coefs <- lasso_regression_model %>%
  predict(type="coefficients", s=optimal_lambda) %>% as.matrix()
lasso_regression_model_coefs[order(lasso_regression_model_coefs,
                                    decreasing=TRUE), ]
```

```
##        (Intercept)          REGIONIX           REGIONX
##       2.476659e+02      5.245002e+01      4.290321e+01
##       REGIONCARAGA             Mun11         REGIONCAR
##       3.745306e+01      3.295658e+01      3.039501e+01
##             cities           REGIONI        REGIONVIII
##       2.888932e+01      2.549179e+01      2.134594e+01
```

```
##                 Mun2            REGIONII            REGIONVI
##         1.559301e+01        1.539264e+01        1.522070e+01
##                 Mun5       WD.AreaArea 6       WD.AreaArea 3
##         1.172404e+01        9.817003e+00        9.523045e+00
##                 surw     nrwpcent_class1           REGIONXI
##         9.381872e+00        7.546674e+00        3.738317e+00
##                 Mun3            conn_log                Mun4
##         3.311140e+00        1.818081e+00        1.040764e+00
##             nrwpcent                 emp conn_p_area_squared
##         7.855176e-01        8.604472e-02        1.944082e-05
##          (Intercept)             vol_nrw              elevar
##         0.000000e+00       -3.637419e-06       -7.365698e-05
##                 conn          conn_p_area                  gw
##        -4.279029e-04       -4.687719e-03       -1.996634e-01
##        WD.AreaArea 4             REGIONV       WD.AreaArea 7
##        -1.623009e+00       -1.674806e+00       -1.740803e+00
##        WD.AreaArea 5            REGIONIV         vol_nrw_log
##        -2.517554e+00       -2.568792e+00       -4.575782e+00
##                 sprw           REGIONIII             coastal
##        -6.275625e+00       -6.855575e+00       -1.068156e+01
##        WD.AreaArea 8       WD.AreaArea 2           REGIONXII
##        -1.534383e+01       -1.960320e+01       -3.893647e+01
##        WD.AreaArea 9           REGIONVII
##        -3.941894e+01       -6.798280e+01
```

In terms of the resulting coefficients, we find that `REGIONIX`, `REGIONX`, `REGIONCARAGA` have the largest positive contribution to `wd_rate`, while `REGIONVII`, `WD.AreaArea 9` and `REGIONXII` have the largest negative contribution to `wd_rate`.

Principal Components Regression

In this section, we fit a principal components regression model to our data.

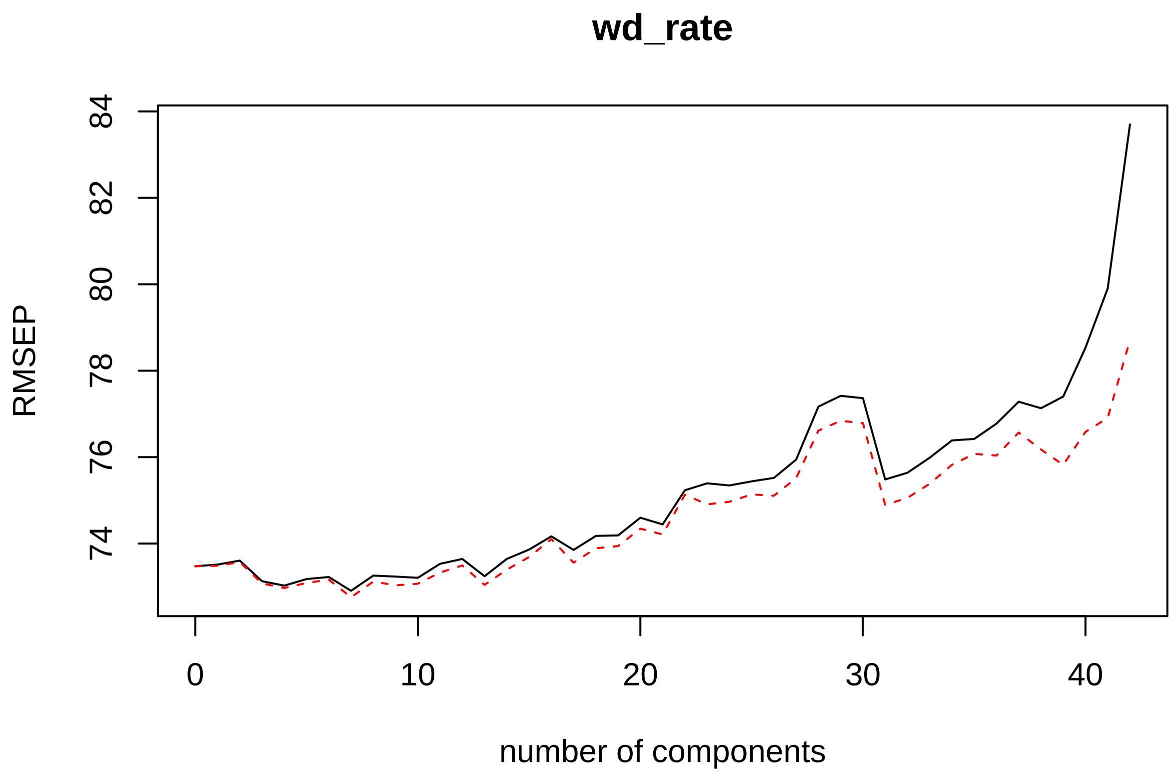
```r
set.seed(1)
pcr_regression_model <- pcr(formulaConstructor(predictors),
                              data=df_train, scale=TRUE, validation="CV")
pcr_regression_model %>% summary
```

```
## Data:    X dimension: 250 42
##  Y dimension: 250 1
## Fit method: svdpc
## Number of components considered: 42
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           73.48    73.51    73.61    73.13    73.03    73.18    73.23
## adjCV        73.48    73.49    73.57    73.08    72.97    73.09    73.17
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       72.91    73.26    73.24     73.21     73.53     73.64     73.24
## adjCV    72.76    73.12    73.03     73.07     73.33     73.49     73.04
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        73.65     73.86     74.17     73.85     74.18     74.19
## adjCV     73.40     73.69     74.10     73.56     73.89     73.94
##        20 comps  21 comps  22 comps  23 comps  24 comps  25 comps
```

```
## CV         74.60      74.44      75.23      75.39      75.34      75.44
## adjCV      74.34      74.21      75.13      74.91      74.97      75.14
##         26 comps  27 comps  28 comps  29 comps  30 comps  31 comps
## CV         75.52      75.94      77.17      77.42      77.37      75.48
## adjCV      75.11      75.52      76.61      76.84      76.79      74.90
##         32 comps  33 comps  34 comps  35 comps  36 comps  37 comps
## CV         75.64      75.99      76.39      76.42      76.77      77.28
## adjCV      75.06      75.38      75.82      76.08      76.04      76.57
##         38 comps  39 comps  40 comps  41 comps  42 comps
## CV         77.13      77.40      78.53      79.90      83.70
## adjCV      76.18      75.82      76.58      76.91      78.71
##
## TRAINING: % variance explained
##           1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X         13.4750  21.5520   27.640   33.353   38.588   43.780   48.566
## wd_rate    0.5465   0.7561    2.424    3.311    3.791    4.089    5.549
##           8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## X          53.086   56.899    60.533    63.832    66.858    69.711
## wd_rate     5.782    7.362     7.603     8.336     8.337     9.128
##           14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## X            72.45    75.105    77.697     80.21     82.47     84.56
## wd_rate       9.37     9.382     9.429     10.79     11.37     11.42
##           20 comps  21 comps  22 comps  23 comps  24 comps  25 comps
## X            86.54     88.32     90.02     91.68     93.29     94.49
## wd_rate      11.46     11.84     11.85     13.57     13.69     13.72
##           26 comps  27 comps  28 comps  29 comps  30 comps  31 comps
## X            95.64     96.75     97.67     98.33     98.81     99.19
## wd_rate      14.33     14.77     15.49     15.69     16.07     20.08
##           32 comps  33 comps  34 comps  35 comps  36 comps  37 comps
## X            99.46     99.63     99.74     99.83     99.92    100.00
## wd_rate      20.08     20.08     20.08     20.13     21.52     21.56
##           38 comps  39 comps  40 comps  41 comps  42 comps
## X           100.00     100.0    100.00    100.00    100.00
## wd_rate      21.75      23.1     23.36     23.41     23.53
```

We create a validation plot for our model to check the best number of components.

```r
validationplot(pcr_regression_model, val.type="RMSEP")
```



By plotting the validation plot over the number of components, we find that we can have minimal CV RMSEP at around 7 components.

Checking the RMSE using 7 as the number of components:

```
predictions <- predict(pcr_regression_model, df_test, ncomp=7) %>% as.vector()
sqrt(mean((predictions - y_test)^2)) / (mean(y_test))
```

```
## [1] 0.2668064
```

Based on our selected PCR model, we achieved 0.2668064 test RMSE.

Classification Problem

Ridge Regression

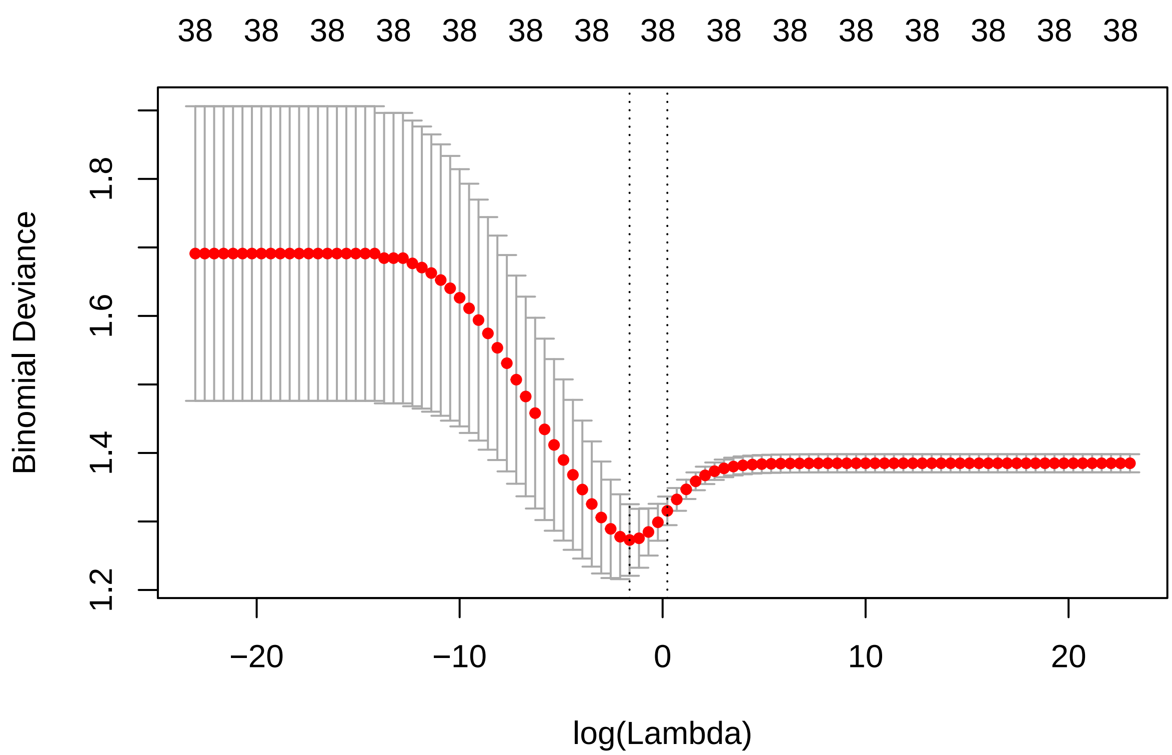
Similar in the Regression Problem, we will also created a `formulaConstructor` for classification.

```
formulaConstructor_c <- function(predictors) {
  predictors %>% paste(collapse=" + ") %>% paste("nrwpcent_class ~", .) %>% as.formula()
}

grid <- 10^seq(10, -10, length=100)

predictors_c <- df_train %>%
  select(-c(wd_rate, wd_rate_log, vol_nrw, vol_nrw_log, nrwpcent,
            nrwpcent_class)) %>% names()

x_train_c <- model.matrix(formulaConstructor_c(predictors_c), df_train)
```

```r
y_train_c <- df_train$nrwpcent_class

x_test_c <- model.matrix(formulaConstructor_c(predictors_c), df_test)
y_test_c <- df_test$nrwpcent_class
set.seed(100)
```

We again find the best lambda for our model:

```r
cv_ridge_classification_model <- cv.glmnet(x_train_c, y_train_c,
                                           alpha=0, lambda = grid, family="binomial")
plot(cv_ridge_classification_model)
```



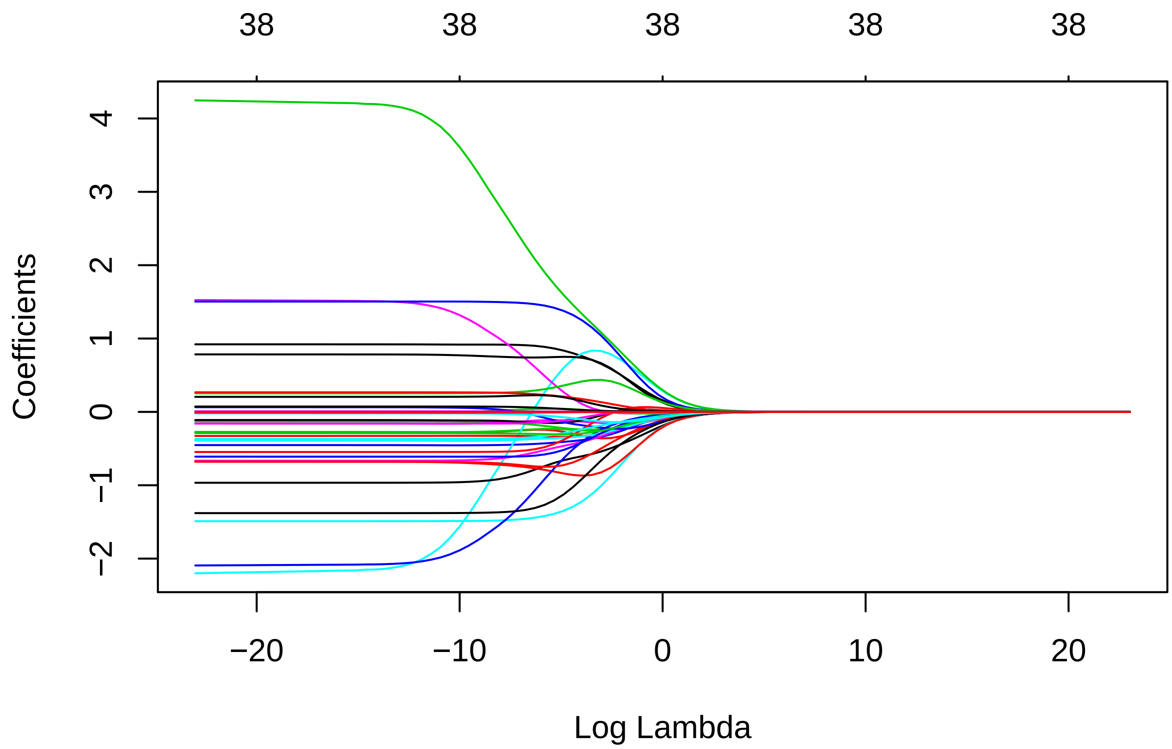
We check the best lambda:

```r
cv_ridge_classification_model$lambda.min
```

```
## [1] 0.1963041
```

Based on the above plot, we find that the optimal value of λ is 0.1963041. Checking the number of coefficients against the log of lambda:

```r
plot(cv_ridge_classification_model$glmnet.fit, "lambda", label=FALSE)
```

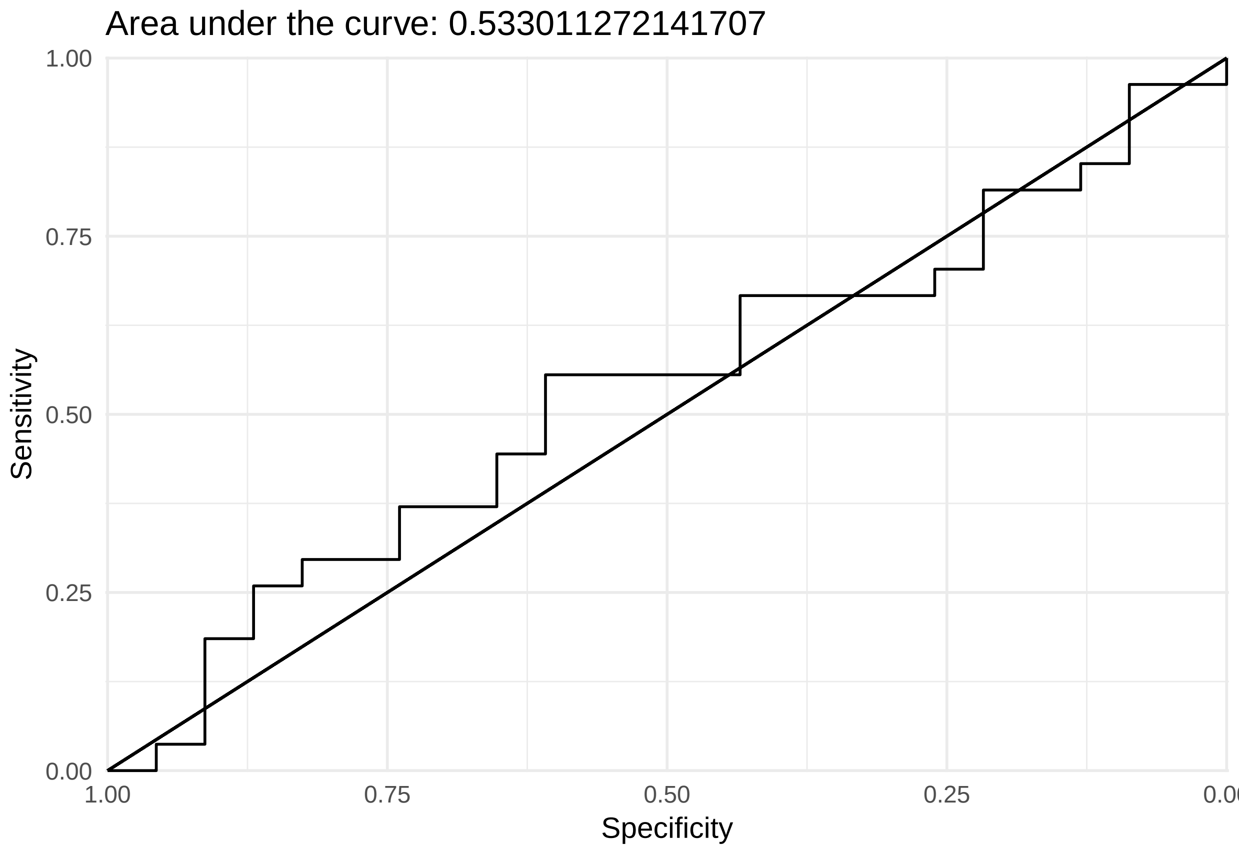


We now evaluate our ridge classification model using the best lambda:

```
# Fit ridge classification model with optimal lambda
optimal_lambda_c <- cv_ridge_classification_model$lambda.min
ridge_classification_model <- glmnet(x_train_c, y_train_c, alpha=0,
                                     lambda = optimal_lambda_c, family = "binomial")
```

To evaluate the performance of our model, we created a helper function that will plot the AUC of out model against the test set:

```
AUCplotter <- function (classifier){
cbind(rev(classifier$specificities), rev(classifier$sensitivities)) %>%
  as.data.frame() %>%
  rename('Specificity'=V1, 'Sensitivity'=V2) %>%
  ggplot(aes(x=Specificity, y=Sensitivity)) +
  geom_segment(aes(x = 0, y = 1, xend = 1,yend = 0), alpha = 0.5)  +
  geom_step() +
  scale_x_reverse(name = "Specificity",limits = c(1,0), expand = c(0.001,0.001)) +
  scale_y_continuous(name = "Sensitivity", limits = c(0,1), expand = c(0.001, 0.001)) +
  labs(title=paste("Area under the curve:", classifier$auc[1], sep=" ")) +
  theme_minimal()
}

predictions <- ridge_classification_model %>% predict(x_test_c) %>% as.vector()
roc_ridge <- roc(y_test_c, predictions)
AUCplotter(roc_ridge)
```



Here, we see that the AUC is 0.533. We also evaluate its accuracy:

```
confusionmatrix_creator <- function(model, x_test, y_test) {
  predicted_probabilities <- model %>% predict(x_test)
  predicted_probabilities[predicted_probabilities > 0.5] <- 1
  predicted_probabilities[predicted_probabilities <= 0.5] <- 0
  predicted_probabilities <- predicted_probabilities %>% as.vector() %>% as.factor
  confusionMatrix(data=predicted_probabilities, reference = as.factor(y_test))
}

confusionmatrix_creator(ridge_classification_model, x_test_c, y_test_c)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 15 18
##          1  8  9
##
##                Accuracy : 0.48
##                  95% CI : (0.3366, 0.6258)
##     No Information Rate : 0.54
##     P-Value [Acc > NIR] : 0.83968
##
##                   Kappa : -0.014
##  Mcnemar's Test P-Value : 0.07756
##
```

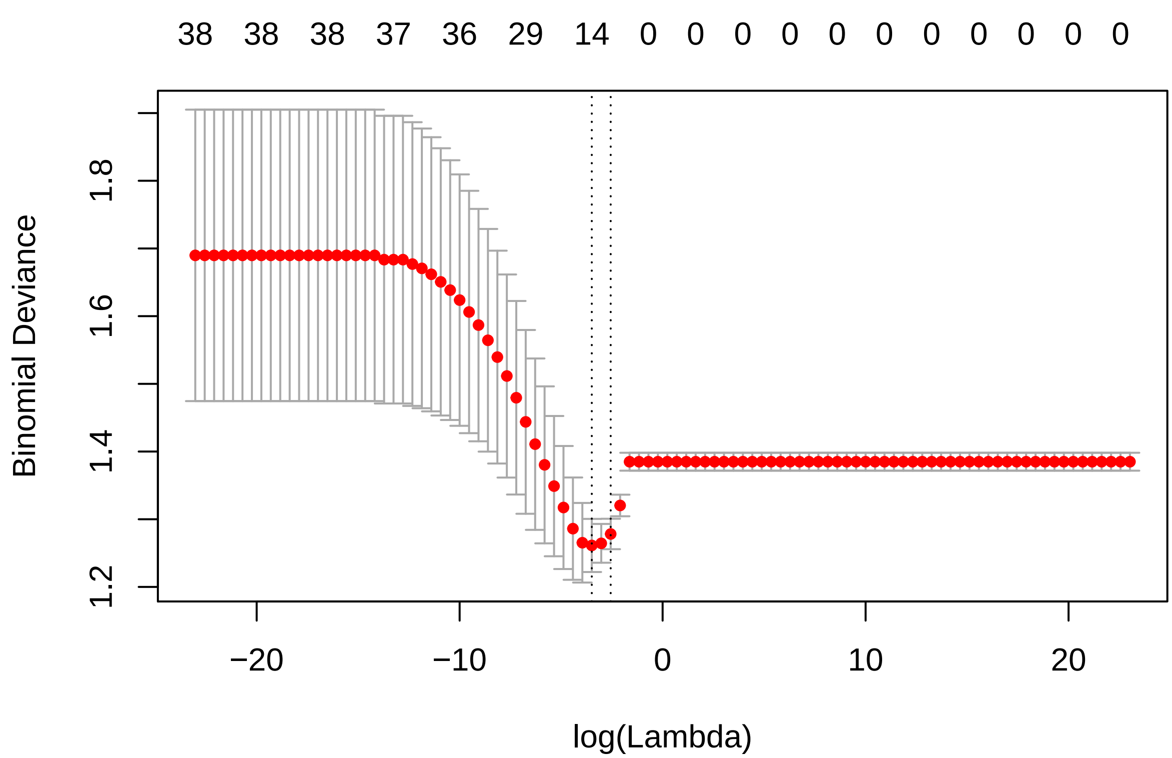
```
##             Sensitivity : 0.6522
##             Specificity : 0.3333
##          Pos Pred Value : 0.4545
##          Neg Pred Value : 0.5294
##              Prevalence : 0.4600
##          Detection Rate : 0.3000
##    Detection Prevalence : 0.6600
##       Balanced Accuracy : 0.4928
##
##        'Positive' Class : 0
##
```

Unfortunately, the accuracy of the Ridge Classification Model is only 48%.

Lasso Regression

We now create a classification model using Lasso Regression. Again, we will just set alpha = 1. We use cross validation again to find an appropriate lambda for our model:

```
set.seed(100)
cv_lasso_classification_model <- cv.glmnet(x_train_c, y_train_c,
                                           lambda = grid, alpha=1, family="binomial")
plot(cv_lasso_classification_model)
```



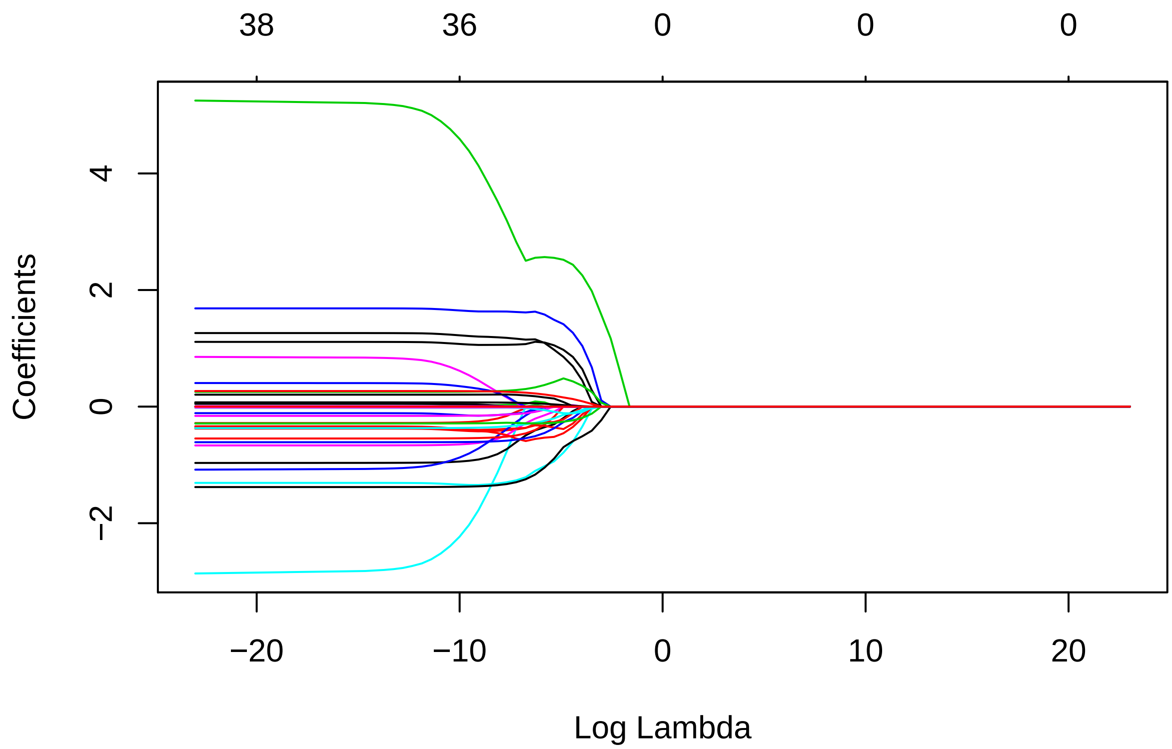
Check the minimim value we want:

```
cv_lasso_classification_model$lambda.min
```

```
## [1] 0.03053856
```

It appears that 0.03053856 is the best lambda for the Lasso model. Checking the coefficients vs. Log Lambda:

```
plot(cv_lasso_classification_model$glmnet.fit, "lambda", label=FALSE)
```

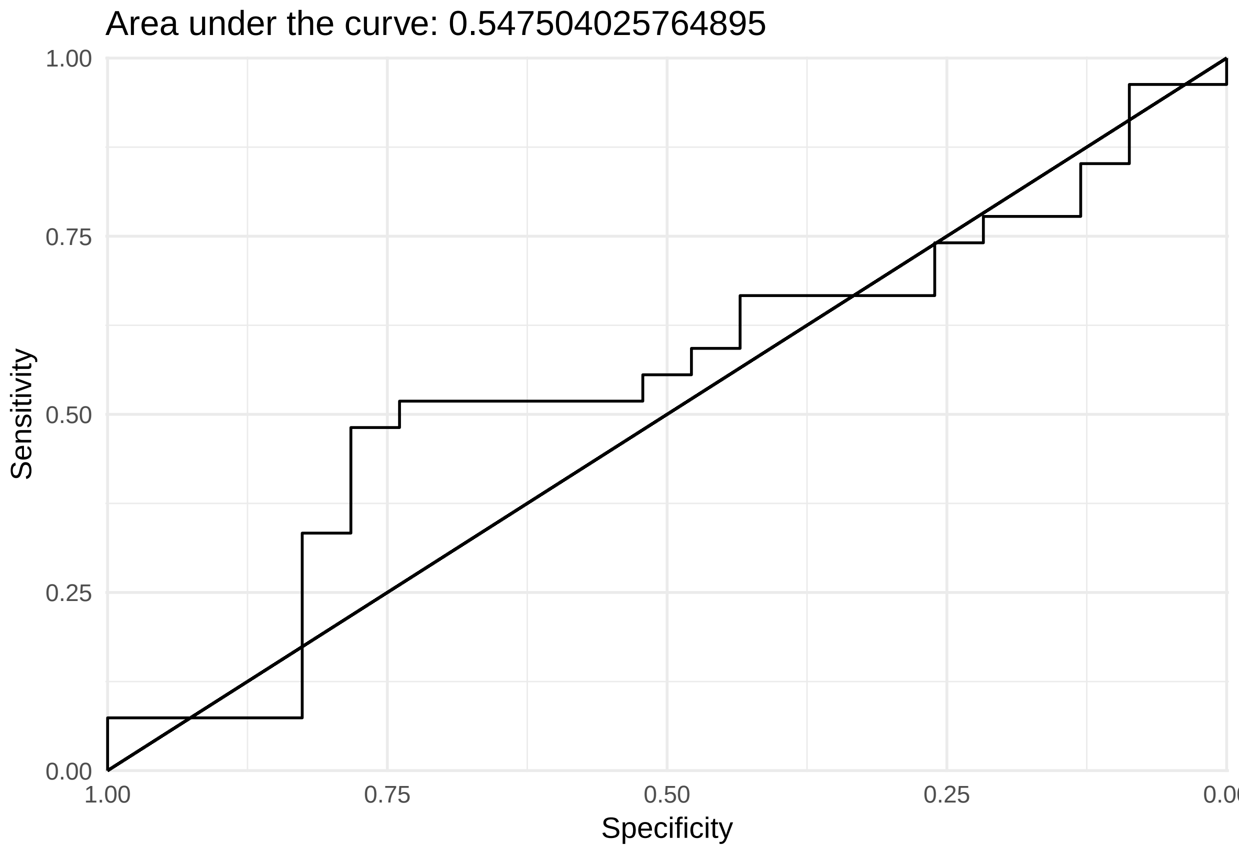


Using our optimum lambda, we create our model.

```
optimal_lambda_c <- cv_lasso_classification_model$lambda.min
lasso_classification_model <- glmnet(x_train_c, y_train_c,
                                     lambda = optimal_lambda_c,
                                     alpha = 1, family = "binomial")
```

We now evaluate this model, checking the AUC:

```
predictions <- lasso_classification_model %>% predict(x_test_c) %>% as.vector()
roc_lasso <- roc(y_test_c, predictions)
AUCplotter(roc_lasso)
```



The AUC for our Lasso Classification Model is 0.548. We now check the accuracy of our model:

```
confusionmatrix_creator(lasso_classification_model, x_test_c, y_test_c)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 17 18
##          1  6  9
##
##                Accuracy : 0.52
##                  95% CI : (0.3742, 0.6634)
##     No Information Rate : 0.54
##     P-Value [Acc > NIR] : 0.66573
##
##                   Kappa : 0.0698
##  Mcnemar's Test P-Value : 0.02474
##
##             Sensitivity : 0.7391
##             Specificity : 0.3333
##          Pos Pred Value : 0.4857
##          Neg Pred Value : 0.6000
##              Prevalence : 0.4600
##          Detection Rate : 0.3400
##    Detection Prevalence : 0.7000
##       Balanced Accuracy : 0.5362
```

```
##
##        'Positive' Class : 0
##
```

Here, we see a better accuracy at 52%, compared to our ridge regression model.

Principal Components Regression

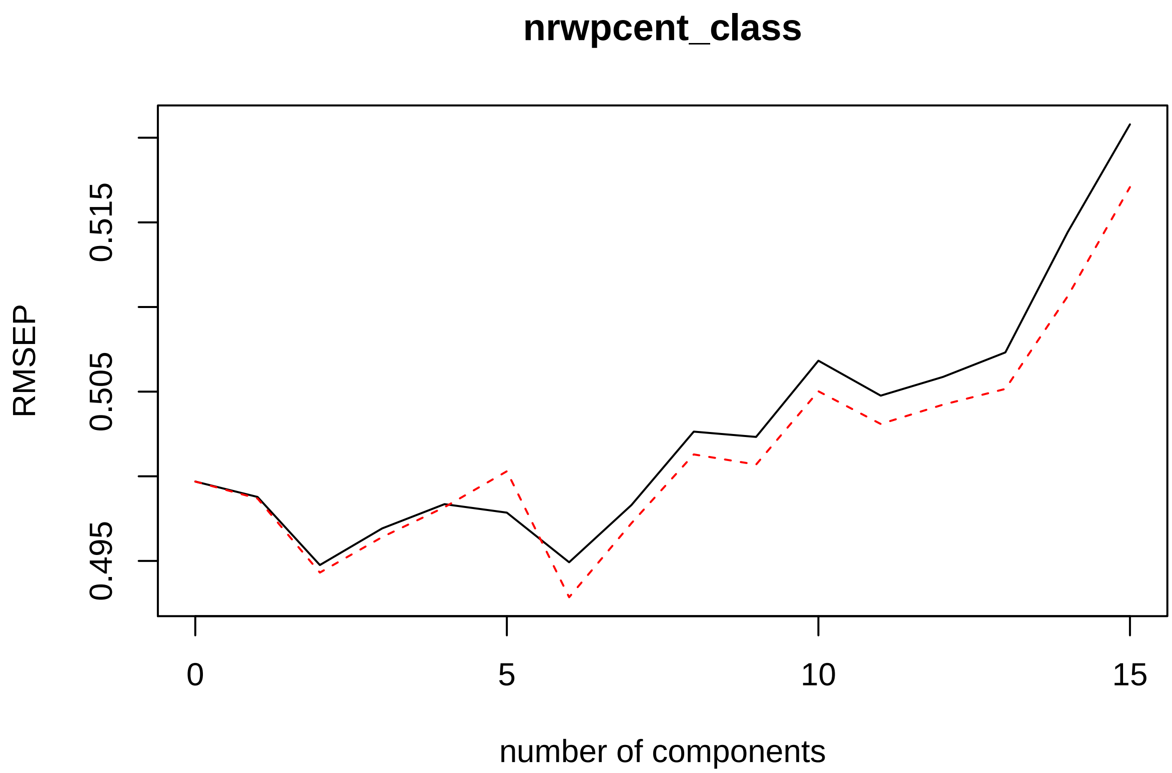
Lastly, we create Principal Components Regression Classifier. We remove some of the components in our df_train based on previous work. We then use it to create our model:

```
predictors_pcr <- df_train %>%
  select(-c(wd_rate, wd_rate_log, vol_nrw, vol_nrw_log, nrwpcent,
            nrwpcent_class, REGION, WD.Area, Mun1)) %>% names()

set.seed(1)
pcr_classification_model <- pcr(formulaConstructor_c(predictors_pcr),
                                data=df_dummies_train, scale=TRUE,
                                validation="CV", family = "binomial")

pcr_classification_model %>% summary
```

```
## Data:    X dimension: 250 15
##  Y dimension: 250 1
## Fit method: svdpc
## Number of components considered: 15
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV          0.4997   0.4988   0.4948   0.4969   0.4984   0.4979   0.4949
## adjCV       0.4997   0.4987   0.4943   0.4964   0.4982   0.5003   0.4929
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV      0.4983   0.5026   0.5023    0.5068    0.5048    0.5059    0.5073
## adjCV   0.4972   0.5013   0.5007    0.5050    0.5031    0.5042    0.5052
##        14 comps  15 comps
## CV       0.5144    0.5208
## adjCV    0.5106    0.5171
##
## TRAINING: % variance explained
##                 1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## X               28.8129    39.92   47.957   55.436   62.222   68.853
## nrwpcent_class   0.2297     3.33    3.664    3.664    3.868    6.806
##                 7 comps  8 comps  9 comps  10 comps  11 comps  12 comps
## X                74.805   80.694    85.91    90.787    94.492    97.189
## nrwpcent_class    6.862    7.138     7.73     7.744     8.279     8.467
##                 13 comps  14 comps  15 comps
## X                  99.37     99.74    100.00
## nrwpcent_class     10.12     13.04     13.15
```

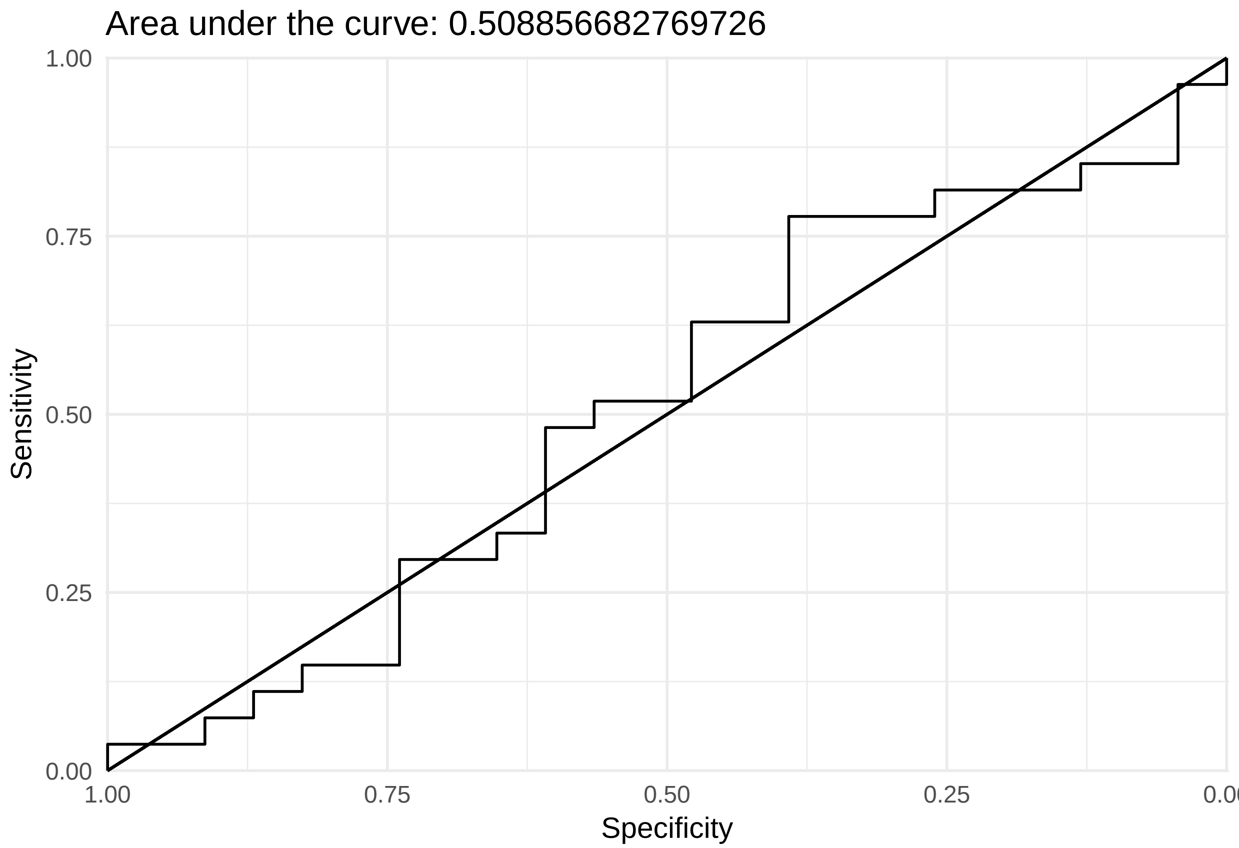
To better visualize the best number of components, we use a validation plot:

```
validationplot(pcr_classification_model, val.type="RMSEP")
```



Here we see that at around 6 number of components would be the best for our model. Using this, we fine tune our model and set the number of components to 6. We then evaluate its performance by checking its AUC:

```
predictions <- predict(pcr_classification_model, df_dummies_test,
                       ncomp=6) %>% as.vector()
roc_pcr <- roc(df_dummies_test$nrwpcent_class, predictions)
AUCplotter(roc_pcr)
```



Here we see an AUC of 0.509. We check the accuracy of our model:

```
predicted_probabilities <- pcr_classification_model %>%
predict(df_dummies_test, ncomp = 6)
predicted_probabilities[predicted_probabilities > 0.5] <- 1
predicted_probabilities[predicted_probabilities <= 0.5] <- 0
predicted_probabilities <- predicted_probabilities %>%
  as.vector() %>% as.factor
confusionMatrix(data=predicted_probabilities,
                reference = as.factor(df_dummies_test$nrwpcent_class))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  9  9
##          1 14 18
##
##                Accuracy : 0.54
##                  95% CI : (0.3932, 0.6819)
##     No Information Rate : 0.54
##     P-Value [Acc > NIR] : 0.5578
##
##                   Kappa : 0.0589
##  Mcnemar's Test P-Value : 0.4042
##
##             Sensitivity : 0.3913
```

```
##              Specificity : 0.6667
##           Pos Pred Value : 0.5000
##           Neg Pred Value : 0.5625
##               Prevalence : 0.4600
##           Detection Rate : 0.1800
##     Detection Prevalence : 0.3600
##        Balanced Accuracy : 0.5290
##
##         'Positive' Class : 0
##
```

So far, the Principal Components Classifier has the highest accuracy.

Conclusions and Recommendations

Summary of Regression Models

For the regression problem, we have the following RMSE metrics:

Model	RMSE
Ridge Regression	0.267
Lasso Regression	0.253
Principal Components Regression	0.268

For the classification problem, we have the following AUC and test accuracy metrics:

Model	AUC	Accuracy
Ridge Regression	0.533	48%
Lasso Regression	0.545	52%
Principal Components Regression	0.509	54%

Based on test RMSE, we found the lasso model to have the best performance for the regression problem. For the classification problem, the best model is the Principal Components Regression Classifier with an accuracy of 54%.