Machine Learning block 1 Lab2 Group A13

Group A13: Arash Haratian, Connor Turner, Yi Hung Chen

2022-11-30

Statement of Contribution: *Assignment 1 was contributed by Yi Hung, Assignment 2 was contributed by Arash, and Assignment 3 was contributed by Connor. Each assignment was then discussed in detail with the group before piecing together each secton of the final report.*

Assignment 1. Explicit regularization

For this assignment, we are given data "tecator.csv" that contains the results of study aimed to investigate whether a near infrared absorbance spectrum can be used to predict the fat content of samples of meat.

Q1. We begin by divide data randomly to train and test set(50/50). We then use the training data to train a linear regression model with all the absorbance characteristics (Channels) as features. Below is the underlying probabilistic model of linear regression

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_{100} x_{100} + \epsilon$$
$$y|x \sim N(\theta^T x, \sigma^2)$$

where

$$y = Fat$$
$$\theta = \theta_0, ..., \theta_{100}$$
$$x = \{1, Channel_1, Channel_2, ..., Channel_{100}\}$$
$$\epsilon \ is \ the \ error \ term$$

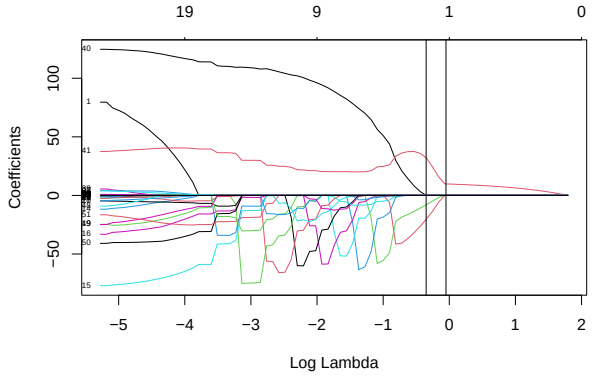
We estimate mean squared errors(MSE) for both training data and testing data. We obtain 0.0057091 for training data and 722.4294193 for testing data. The large MSE on testing data indicates the linear regression model is overfitting. To improve this, we are going to use Lasso and Ridge regression in this assignment.

Q2. We then switch to use LASSO regression model. Which the cost function that should be optimized is.

$$\hat{\theta^{lasso}} = \{\frac{1}{n}\sum_{i=1}^{n}(y_i - \theta_0 - \sum_{j=1}^{p}\theta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}|\theta_j|\}$$

Q3. We fit the LASSO regression model to the training data. The dependency of the regression coefficients on the $\log(\lambda)$ is shown below in Figure 1. As the plot shown, the LASSO will drop some coefficients as λ increase (set the values to 0). Interestingly, when λ change, some coefficient will reappear.

Figure 1:
Dependence of Lasso Regression coefficients on log(lambda)

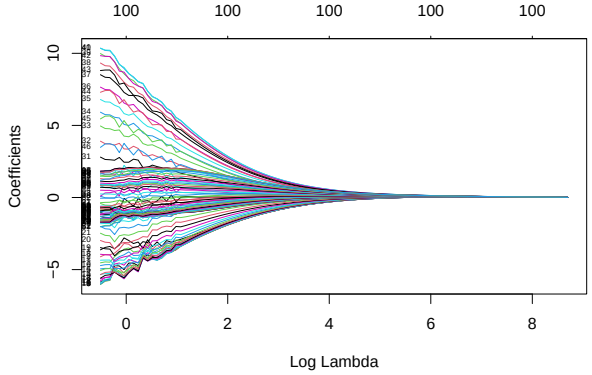


If we want to select a model with only three features, we can choose λ around 0.9512294 to 0.7046881.(Note: The range is choosen by dierectly observed with the plot(vertical lines). Additionally, by using default lambdas in glmnet, we calculated $\lambda = 0.8530452, 0.777263, 0.7082131$ will gives coefficient for only three features. However, as the plot shown, the lambdas should be in a range not district values.)

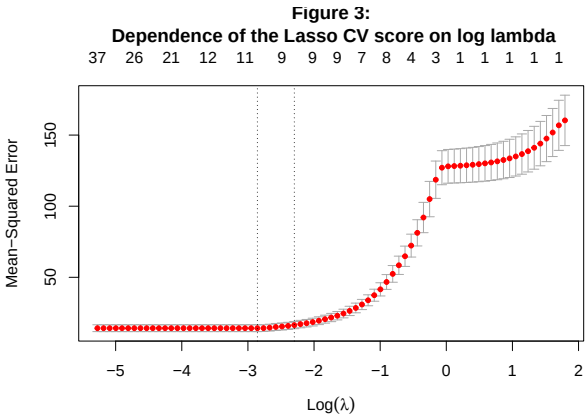
Q4. We then repeat the privious step, but instead of using Lasso Regression, we use Ridge Regression to compare both method.

As the Figure 2 shown. Compare to Lasso Regression, the penalty term of Ridge Regression will not "collapse" to zero. Also, the coefficient value of Ridge Regression scale slower when lambda increase compared to Lasso Regression.

Figure 2:
Dependence of Ridge Regression coefficients on log(lambda)



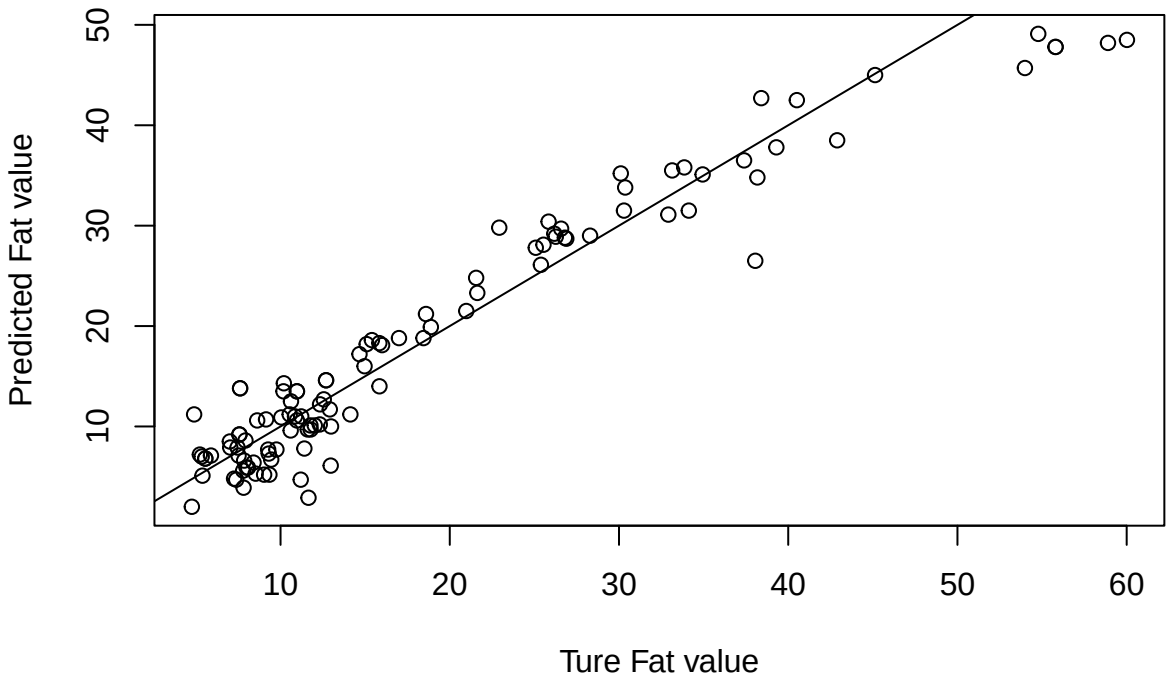
Q5. We use cv.glmnet function to do cross-validation for Lasso model. We do not specify how many folds being use, instead we use the default number of folds to compute. As the Figure 3 shown, the MSE start increasing dramatically as $\log(\lambda)$ approach -2, which means the the model become less usable, this might result from too many coefficient being ditched by Lasso regression as lambda increase.



We obtain the best λ =0.0574453 with 7 variables chosen(Channel13, Channel14, Channel15, Channel16, Channel40, Channel41, Channel51). According to Figure 3, the optimal λ ($\log(\lambda)$ =-2.8569213) is not statistically significantly better than $\log\lambda$ = -4, as they have similar MSE.

Finally, we plot the scatter plot (Figure 4) with true "Fat" value from test data on the X-axis and predict value on the Y-axis. We can observed that the prediction is much better compare to linear regression(MSE = 13.2997953). Note: We also include a line x=y which indicate where the perfect prediction should lay on.

Figure 4:Prediction of using Optimal Lasso on True test data



Assignment 2

```
## Attaching Packages
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x tidyr::expand() masks Matrix::expand()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x tidyr::pack()   masks Matrix::pack()
## x tidyr::unpack() masks Matrix::unpack()
```

```
library(tree)
```

In this assignment we are given the data related with direct marketing campaigns of a Portuguese banking institution, and we have to use decision tree model to classify the target variable `y` using 15 features. The target variable has two levels: `yes` and `no`.

```
bank_full <- read.csv2("./data/bank-full.csv")
bank_full <- bank_full %>%
    select(!duration) %>%
    mutate(across(where(is.character), as.factor))

n <- nrow(bank_full)
set.seed(12345)
train_idx <- sample(seq_len(n), floor(n * 0.4))
train_data <- bank_full[train_idx, ]
remainder_idx <- setdiff(seq_len(n), train_idx)
set.seed(12345)
valid_idx <- sample(remainder_idx, floor(n * 0.3))
valid_data <- bank_full[valid_idx, ]
test_idx <- setdiff(remainder_idx, valid_idx)
test_data <- bank_full[test_idx, ]
```

We begin by splitting data 40/30/30 into training, validation, and testing sets. We then use the training data to train 3 different decision trees with 3 different settings:

1. First Decision Tree:

The first model is a decision tree with default settings of the `tree::tree()`. The default values are as follows: - `mincut = 5` - `minsize = 10` - `mindev = 0.01`

```
tree_default <- tree(y ~ ., train_data)
y_hat_train <- predict(tree_default, train_data, type = "class")
cm_train <- table(train_data$y, y_hat_train)
error_train_a <- 1 - (sum(diag(cm_train))/nrow(train_data))
# mean(y_hat_train != train_data$y)
```

```
y_hat_train <- predict(tree_default, valid_data, type = "class")
cm_valid <- table(valid_data$y, y_hat_train)
error_test_a <- 1 - (sum(diag(cm_valid))/nrow(valid_data))
```

The training and test errors for this model are 0.1048 and 0.1093 respectively.

2. Second Decision Tree: The second model is a decision tree with a constraint of the size of each node; In fact, the smallest allowed node size should be equal to 7000. To set this setting, we should pass `tree::tree.control(minsize = 7000)` to the `control` argument of the `tree::tree()`.

```
tree_minsize <- tree(y ~ ., train_data, control = tree.control(nrow(train_data),
    minsize = 7000))
y_hat_train <- predict(tree_minsize, train_data, type = "class")
cm_train <- table(train_data$y, y_hat_train)
error_train_b <- 1 - (sum(diag(cm_train))/nrow(train_data))
# mean(y_hat_train != train_data$y)
y_hat_train <- predict(tree_minsize, valid_data, type = "class")
cm_valid <- table(valid_data$y, y_hat_train)
error_test_b <- 1 - (sum(diag(cm_valid))/nrow(valid_data))
```

The training and test errors for the second model are 0.1048 and 0.1093 respectively.

3. Third Decision Tree:

The third model is a decision we want to set the minimum deviance to 0.0005, and it can be done in the same way as the second model, by passing `tree::tree.control(mindev = 0.0005)` to the `control` argument of the `tree::tree()`.

```
tree_mindev <- tree(y ~ ., train_data, control = tree.control(nrow(train_data),
    mindev = 5e-04))
y_hat_train <- predict(tree_mindev, train_data, type = "class")
cm_train <- table(train_data$y, y_hat_train)
error_train_c <- 1 - (sum(diag(cm_train))/nrow(train_data))
# mean(y_hat_train != train_data$y)
y_hat_train <- predict(tree_mindev, valid_data, type = "class")
cm_valid <- table(valid_data$y, y_hat_train)
error_test_c <- 1 - (sum(diag(cm_valid))/nrow(valid_data))
```

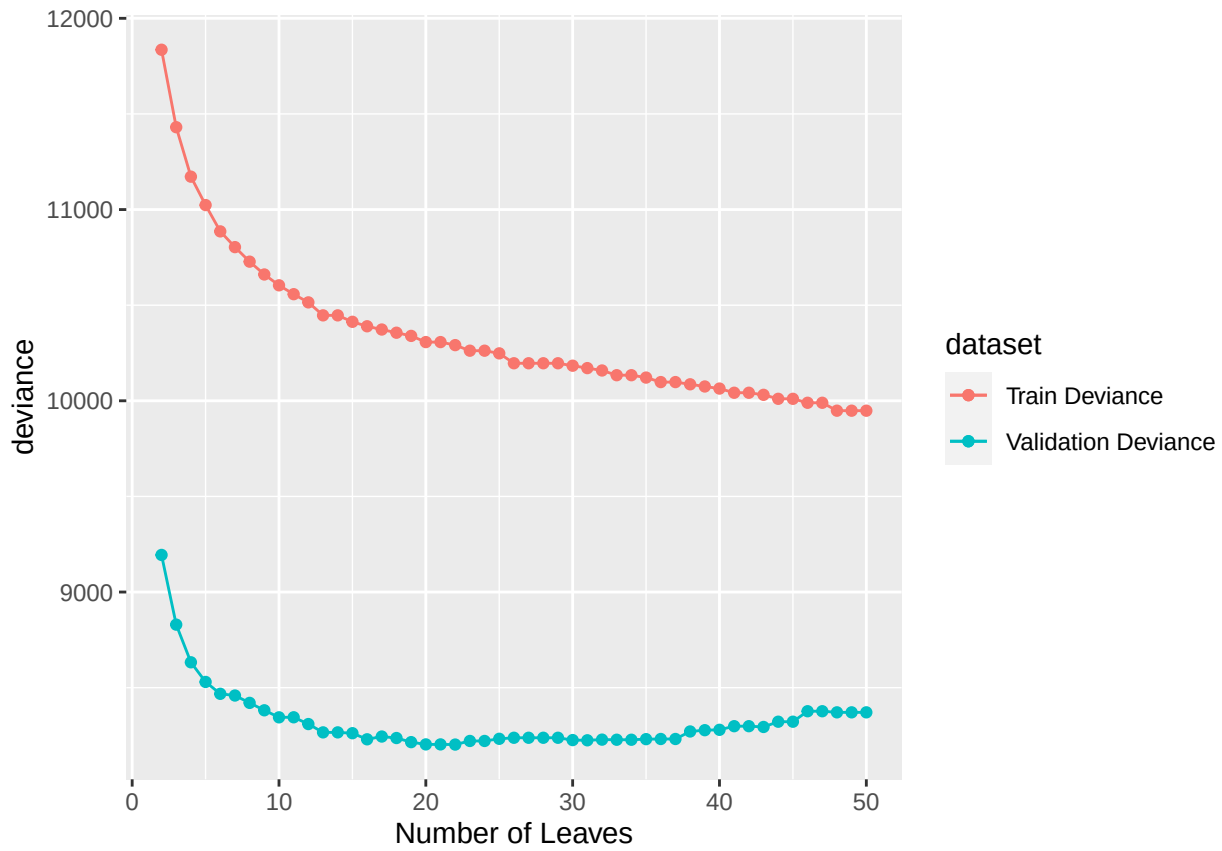
The training and test errors for the third model are 0.094 and 0.1119 respectively.

different values	First Model	Second Model	Third Model
Size of Trees	6	5	122
Number of feature used	4	3	13
Training Error	0.1048	0.1048	0.094
Testing Error	0.1093	0.1093	0.1119

Based on the information in the table above, first model and second model have the same error rate. However, both the number of terminal nodes and number of feature used to split for the second model is 1 less that the first model. In our opinion the second model is more proper for interpreting the structure of the tree, but for predicting new values, the first model may preform better as it is more complex compare to the second model. The third model is overfitted on the training data and it is too complex; as the result, it may preform poorly for predicting new values.

Now we can prune the more complex model, in this case the third model(minimum deviance to 0.0005), to reach to a more optimal model. By doing so, we may end up with a model which captures the general structure better than other models while it is not overfitted on the training dataset. In other words, by pruning a fully-grown tree, we are mitigating the overfitting (or high variance) problem. The following figure shows the dependence of deviances for the training and the validation data in the number of leaves:

```
train_dev <- vector("numeric", 50)
valid_dev <- vector("numeric", 50)

for (num_leaves in 2:50) {
    prune_tree <- prune.tree(tree_mindev, best = num_leaves)
    valid_tree <- predict(prune_tree, newdata = valid_data,
        type = "tree")
    train_dev[num_leaves] <- deviance(prune_tree)
    valid_dev[num_leaves] <- deviance(valid_tree)
}

data.frame(num_of_leaves = 2:50, train = train_dev[2:50],
    valid = valid_dev[2:50]) %>%
    ggplot() + geom_line(aes(num_of_leaves, train, color = "Train Deviance")) +
    geom_point(aes(num_of_leaves, train, color = "Train Deviance")) +
    geom_line(aes(num_of_leaves, valid, color = "Validation Deviance")) +
    geom_point(aes(num_of_leaves, valid, color = "Validation Deviance")) +
    labs(x = "Number of Leaves", y = "deviance", color = "dataset")
```



```
best_num_leaves <- which.min(valid_dev[2:50]) + 1
optimal_tree <- prune.tree(tree_mindev, best = best_num_leaves)
```

As it is evidence in the plot, the deviance of the training data set is decreasing by increasing the number of leaves(complexity), whereas the deviance of the validation data set is decreasing at first but then it starts to increase. This denotes that the model will overfit the by increasing the number of terminal nodes. It worth to note that the deviance of the training data set is higher than validation dataset because deviance is proportional to the number of observations in the dataset.

The best number of the leaves is 22.

The important variables in the optimal tree are as follows: - `poutcome` - `month` - `contact` - `pdays` - `age` - `day` - `balance` - `housing` - `job` Also we can see the the structure of the tree:

```
optimal_tree
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##    1) root 18084 12850.00 no ( 0.88576 0.11424 )
##      2) poutcome: failure,other,unknown 17468 11030.00 no ( 0.90422 0.09578 )
##        4) month: apr,aug,feb,jan,jul,jun,may,nov 16828  9772.00 no ( 0.91520 0.08480 )
##          8) contact: unknown 5130  1599.00 no ( 0.96374 0.03626 )
##           16) month: jul,jun,may 5074  1502.00 no ( 0.96610 0.03390 ) *
##           17) month: apr,aug,feb,jan,nov 56    62.98 no ( 0.75000 0.25000 ) *
##          9) contact: cellular,telephone 11698  7914.00 no ( 0.89391 0.10609 )
##           18) month: aug,jan,jul,may,nov 9284  5503.00 no ( 0.91265 0.08735 )
##             36) pdays < 383.5 9246  5373.00 no ( 0.91510 0.08490 )
##               72) age < 60.5 9097  5107.00 no ( 0.91920 0.08080 )
##                144) day < 27.5 7670  4588.00 no ( 0.91147 0.08853 )
##                  288) age < 29.5 754   637.10 no ( 0.85013 0.14987 )
##                    576) month: jan,jul,may 681   528.00 no ( 0.86931 0.13069 ) *
##                    577) month: aug,nov 73    92.46 no ( 0.67123 0.32877 ) *
##                  289) age > 29.5 6916  3918.00 no ( 0.91816 0.08184 )
##                    578) balance < 1493 5180  2635.00 no ( 0.92973 0.07027 )
##                     1156) month: aug,jul,may,nov 5164  2596.00 no ( 0.93087 0.06913 ) *
##                     1157) month: jan 16    21.93 no ( 0.56250 0.43750 ) *
##                    579) balance > 1493 1736  1249.00 no ( 0.88364 0.11636 ) *
##                145) day > 27.5 1427   472.40 no ( 0.96076 0.03924 )
##                  290) pdays < 184.5 1308   462.50 no ( 0.95719 0.04281 )
##                    580) pdays < 80.5 1277   402.60 no ( 0.96319 0.03681 ) *
##                    581) pdays > 80.5 31    37.35 no ( 0.70968 0.29032 ) *
##                  291) pdays > 184.5 119     0.00 no ( 1.00000 0.00000 ) *
##               73) age > 60.5 149   190.10 no ( 0.66443 0.33557 ) *
##             37) pdays > 383.5 38    47.40 yes ( 0.31579 0.68421 ) *
##           19) month: apr,feb,jun 2414  2262.00 no ( 0.82187 0.17813 )
##             38) housing: no 1123  1321.00 no ( 0.72484 0.27516 )
##               76) day < 9.5 691   668.20 no ( 0.81187 0.18813 )
##                152) month: feb 505   364.20 no ( 0.88317 0.11683 ) *
##                153) month: apr,jun 186   247.30 no ( 0.61828 0.38172 ) *
##               77) day > 9.5 432   586.10 no ( 0.58565 0.41435 ) *
##             39) housing: yes 1291   803.20 no ( 0.90627 0.09373 )
##               78) day < 20.5 1210   655.90 no ( 0.92314 0.07686 )
##                156) month: apr,feb 1154   565.70 no ( 0.93328 0.06672 ) *
```

```
##                157) month: jun 56    67.01 no ( 0.71429 0.28571 ) *
##               79) day > 20.5 81   104.40 no ( 0.65432 0.34568 ) *
##        5) month: dec,mar,oct,sep 640   852.70 no ( 0.61562 0.38438 ) *
##      3) poutcome: success 616   806.40 yes ( 0.36201 0.63799 )
##        6) pdays < 94.5 170   185.50 yes ( 0.23529 0.76471 ) *
##        7) pdays > 94.5 446   603.90 yes ( 0.41031 0.58969 )
##         14) job: admin.,blue-collar,entrepreneur,services,technician 213   295.20 no ( 0.50704 0.492
##         15) job: housemaid,management,retired,self-employed,student,unemployed,unknown 233   292.80 y
```

Based on the structure of the tree, the first variable that has been used to split the data, is `poutcome` which is the outcome of the previous marketing campaign. Also it is worth to note that the variable `month` (last contact month of year) has been used frequently for splitting the data.

For evaluating the performance of the optimal tree we can check the confusion matrix, accuracy and F1-score for the testing data. F1-score is better metric to measure the performance of the model as the target variable y is unbalanced.

```
y_hat <- predict(optimal_tree, newdata = test_data, type = "class")

cm_test <- table(test_data$y, y_hat)
acc <- sum(diag(cm_test))/nrow(test_data)

TP <- cm_test[2, 2]
FP <- cm_test[1, 2]
FN <- cm_test[2, 1]

F1_score <- TP/(TP + 0.5 * (FP + FN))

knitr::kable(cm_test, label = "the confusion matrix of optimal tree on test data")
```

	no	yes
no	11872	107
yes	1371	214

The accuracy is 0.891 which indicates that model is very accurate in predicting new values. However, as mentioned before, since the target variable has mostly `no` values, a model which predicts all the values as `no` will yield a high accuracy as well. A good model is a model which can predict `yes` more accurately in this dataset. F1-score for this model is 0.224554 which is very low and it indcates that model is not preforming good in predicting true positives.

One way to increase the performance of the model is to use a loss matrix to predict the values based on their probabilities (instead of using 0.5 as threshold, we are using `1/5`). In other words, we are penalizing the model if it predicts positives wrongly. The confusion matrix for the testing data and using the loss matrix is as follows:

```
y_hat <- predict(optimal_tree, newdata = test_data)
y_hat <- as.factor(ifelse(y_hat[, 2]/y_hat[, 1] > 1/5, 1,
    0))


cm_test <- table(test_data$y, y_hat)
acc <- sum(diag(cm_test))/nrow(test_data)
```

```r
TP <- cm_test[2, 2]
FP <- cm_test[1, 2]
FN <- cm_test[2, 1]

F1_score <- TP/(TP + 0.5 * (FP + FN))

knitr::kable(cm_test, label = "the confusion matrix of optimal tree on test data with loss matrix")
```

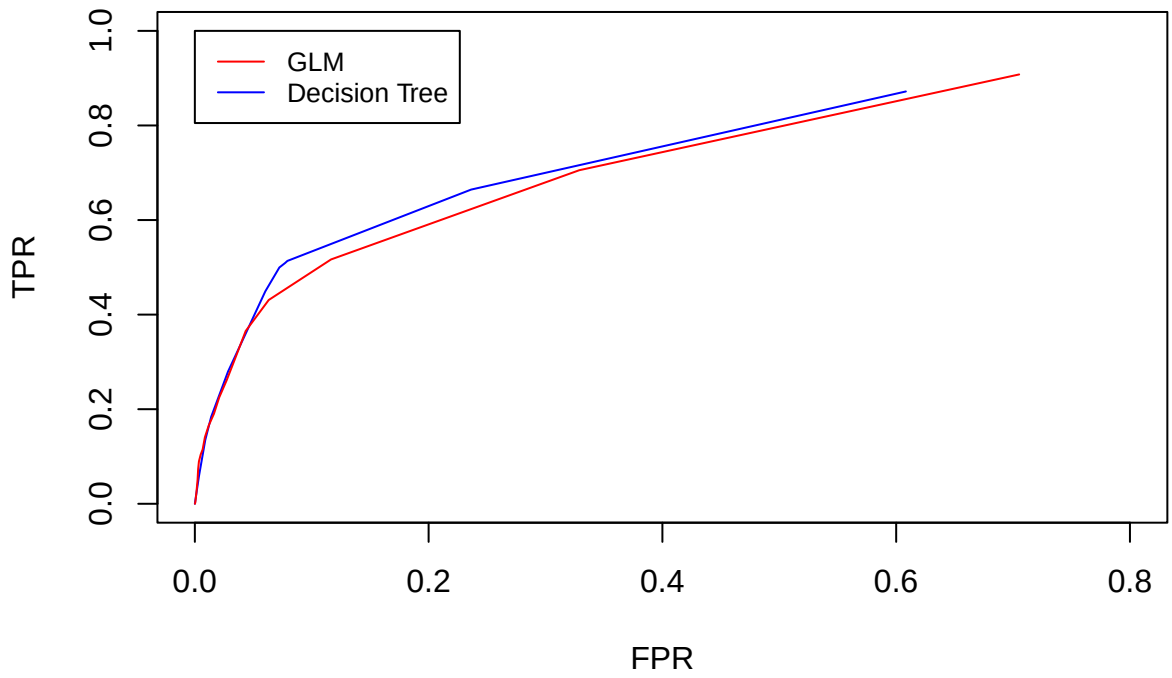
	0	1
no	11030	949
yes	771	814

The new F1-score is 0.4862605 which means now the model predict true positives more accurately although the accuracy has decreased to 0.8732.

One way to compare two different models is to use a ROC curve. Here we want to compare the optimal decision tree with a logistic regression.

```r
y_hat_prob_tree <- predict(optimal_tree, test_data)
y_hat_prob_tree <- y_hat_prob_tree[, "yes"]

glm_model <- glm(y ~ ., family = "binomial", data = train_data)
y_hat_prob_glm <- predict(glm_model, test_data, type = "response")

pis <- seq(0.05, 0.95, by = 0.05)

ROC_tree <- matrix(nrow = length(pis), ncol = 2)
ROC_glm <- matrix(nrow = length(pis), ncol = 2)

for (i in seq_along(pis)) {

    y_hat_tree <- factor(ifelse(y_hat_prob_tree > pis[i],
        "yes", "no"), levels = c("no", "yes"))
    cm_test <- table(test_data$y, y_hat_tree)
    TP <- cm_test[2, 2]
    FP <- cm_test[1, 2]
    ROC_tree[i, 1] <- FP/sum(cm_test[1, ])
    ROC_tree[i, 2] <- TP/sum(cm_test[2, ])

    y_hat_glm <- factor(ifelse(y_hat_prob_glm > pis[i],
        "yes", "no"), levels = c("no", "yes"))
    cm_test <- table(test_data$y, y_hat_glm)
    TP <- cm_test[2, 2]
    FP <- cm_test[1, 2]
    ROC_glm[i, 1] <- FP/sum(cm_test[1, ])
    ROC_glm[i, 2] <- TP/sum(cm_test[2, ])

}

plot(ROC_tree, type = "l", col = "blue", xlim = c(0, 0.8),
    ylim = c(0, 1), xlab = "FPR", ylab = "TPR")
```

```
lines(ROC_glm, col = "red")
legend(0, 1, legend = c("GLM", "Decision Tree"), col = c("red",
    "blue"), lty = 1, cex = 0.8)
```



Based on the ROC curve, both models are almost preform in the same way, but decision tree is slightly better. For the same value or π and same value of False Positive Rate (FPR), the decision tree has higher value of True Positive Rate.

The precision-recall curve is a better choice to compare two models as the classes of target data is imbalanced. The model which has the higher recall for the same value of precision, is more accurate to predict positive values.

Appendix

```
# ===== Assignment 1=====

# ==Q1==
rm(list = ls())
library(glmnet)
# Split and Prepare Data
# ------------------------------------------------
# Split the Data Into Training and Testing Data
# (50/50):
tecator <- read.csv("tecator.csv")
set.seed(12345)
n = nrow(tecator)
id = sample(1:n, floor(n * 0.5))
train = tecator[id, ]
test = tecator[-id, ]

# =====Fit Linear Regression=====
lm_tecator <- lm(formula = Fat ~ . - Protein - Moisture -
    Sample, data = train)

train_fitvalues <- lm_tecator$fitted.values
test_predict <- predict(lm_tecator, test)


MSE_train <- mean(lm_tecator$residuals^2)
MSE_test <- mean((test$Fat - test_predict)^2)


# ==Q2,Q3== =====Lasso===== Make argument in the
# format that glmnet can use (data matrix for x)
x_name <- colnames(tecator)[-1]
x_name <- x_name[-102]
x_name <- x_name[-102]
x_name <- x_name[-101]
x <- data.matrix(train[, x_name])
y <- train$Fat

# Train Lasso
lasso_tecator <- glmnet(x = x, y = y, alpha = 1)

plot(lasso_tecator, xvar = "lambda", label = TRUE, main = "Figure 1:\n Dependence of Lasso Regression c
    abline(v = -0.05) + abline(v = -0.35)

# pick lambda for 3 features
coef_matrix <- as.matrix(coef(lasso_tecator))
coef_matrix <- coef_matrix != 0
coef_matrix <- colSums(coef_matrix)
lamda_index <- which(coef_matrix == 4)  # we use 4 because there is alway intercept with in the matric
lambda_3features <- lasso_tecator$lambda[lamda_index]  # the lamda for 3 features
```

```r
# ==Q4== =====Ridge======
ridge_tecator <- glmnet(x, y, alpha = 0)
plot(ridge_tecator, xvar = "lambda", label = TRUE, main = "Figure 2:\n Dependence of Ridge Regression c

# ==Q5==
cv_lasso_tecator <- cv.glmnet(x = x, y = y, alpha = 1)  #Use cv.glmnet to do cross validation

plot(cv_lasso_tecator, main = "Figure 3:\n Dependence of the Lasso CV score on log lambda\n\n")

best_lambda <- cv_lasso_tecator$lambda.min
best_model <- glmnet(x, y, alpha = 1, lambda = best_lambda)
variables <- coef(best_model)
variables <- as.vector(variables != 0)
coef_index <- which(match(variables, TRUE) == 1)
coef_index <- coef_index[-1]

coef_get_select <- rownames(coef(best_model))[coef_index]

# ===== END Assignment 1=====
```