Case Study on Pima Indians Diabetes Data Set(By Xuan Zhou)

Xuan Zhou

October 29, 2016

1. Introduction

1.1 Background Information

This is a supervised learning problem which we need to make predictions on whether a person is to suffer the diabetes given the 8 features in the dataset.

We can learn following important information from the Data Set Description and ** UPDATE:

- All patients (768 Observations) in this dataset contains are females at least 21 years old of Pima Indian heritage.
- All zero values for the biological variables other than number of times pregnant should be treated as missing values.

1.2 Two Supervised Learning Methods

- Random Forest
- Logistic Regresstion

Key Assumption for Logistic Regresstion: The independent variables should be independent from each other.

1.3 Avoid Overfitting

- Random Forest: use out-of-bag estimate(OOB)
- Logistic Regresstion: use regularization and 10 folds cross-validation.

1.4 Model Assessment

I analyze the models through the model accuracy (1 - classification error rate).

Models are trained on the 80% of data (614 Observations), and test the better model on 20% data (154 Observations).

1.5 Content

There are six main parts to my script as follows:

- Data Preprocessing
- Missing Data Imputation
- Feature Engineering
- Build Models
- Comparison and Conclusion
- Model Investigattion and Improvements

2. Data Preprocessing

2.1 Import Data and Check Missing Values

```
library(mice)
library(randomForest)
library(ggplot2)
library(glmnet)

# Inmport Pima Indians Diabetes Database from UCI Machine Learning Repository
link <- "http://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-di
dataset <- read.table(link, sep = ",", strip.white=TRUE, fill = F)
colnames(dataset) <- c("preg_times", "glucose_test", "blood_press", "tsk_thickness",
                       "serum", "bm_index", "pedigree_fun", "age", "class")
dataset$class <- as.factor(dataset$class)

# Check if there has NA in the dataset
print(all(!is.na(dataset)))
```

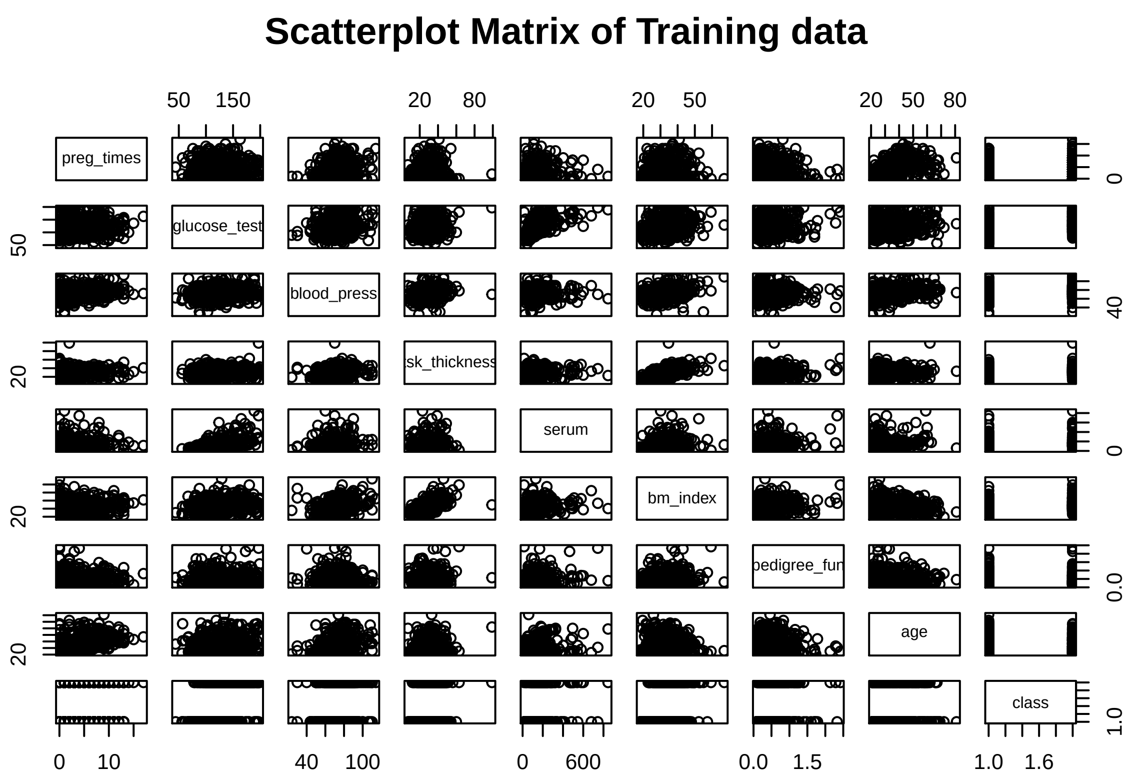
```
## [1] TRUE
```

```
# Treat 0 in the biological variables other than number of times pregnant as missing values
cols_change <- colnames(dataset)[!colnames(dataset) %in% c("preg_times", "class")]
bool_data <- dataset[cols_change] == 0
dataset[cols_change][bool_data] <- NA

# Show the number of missing values of each column
print(apply(bool_data, 2, sum))
```

```
##  glucose_test   blood_press tsk_thickness         serum      bm_index
##             5            35           227           374            11
##  pedigree_fun           age
##             0             0
```

2.2 Analyze the Dataset

```
# Set a random seed
set.seed(123)
# Split the dataset: 80% for trainging and 20% for testing
# trainging: dataset[train,]
# testing: dataset[-(train),]
train <- sample(nrow(dataset), round(0.8*nrow(dataset)), replace = FALSE)

# Show scatterplot matrix on the training data
pairs(~.,data=dataset[train,], main="Scatterplot Matrix of Training data")
```



The scatterplot matrix above shows:

- No obvious high correlation between independent variables.
- No obvious relationship between diastolic blood pressure and diabetes.
- No obvious relationship between age and diabetes.

3. Missing Data Imputation

Because of the small size of the dataset, I want to obtain as much as information from it, so I will not delete either entire observations (rows) or variables (columns) containing missing values right now.

Two options:

- Replacing missing data with sensible values (mean or median) given the distribution of the data.
- Replacing missing data with prediction (Mutiple Imputaion).

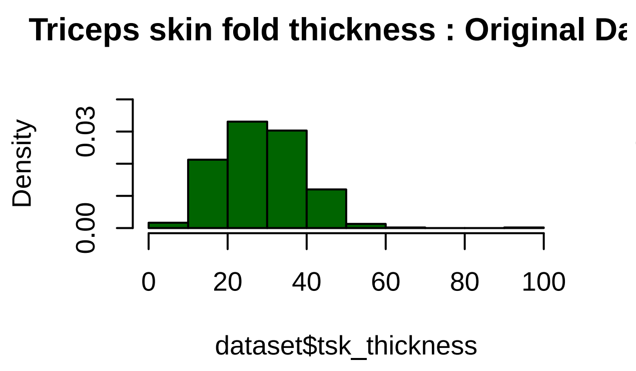
Iâ ll use 1st (median) on the small number of missing values and 2nd (Mutiple Imputaion) on the large number of missing values.

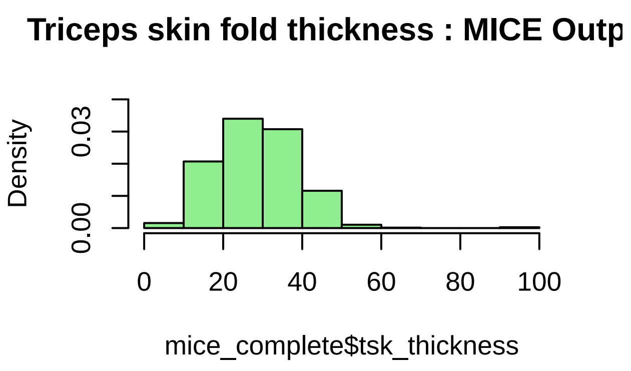
```
# Median value imputation
dataset$glucose_test[is.na(dataset$glucose_test)] <- median(dataset$glucose_test,na.rm = T)
dataset$blood_press[is.na(dataset$blood_press)] <- median(dataset$blood_press,na.rm = T)
dataset$bm_index[is.na(dataset$bm_index)] <- median(dataset$bm_index,na.rm = T)
```

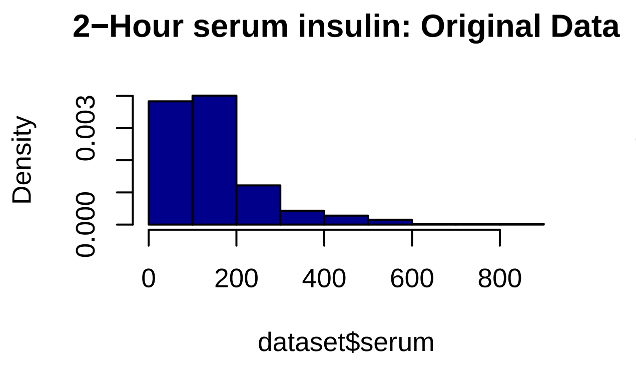
```r
# Multiple imputation
mice_mod <- mice(dataset[, c("tsk_thickness","serum")], method='rf')
```

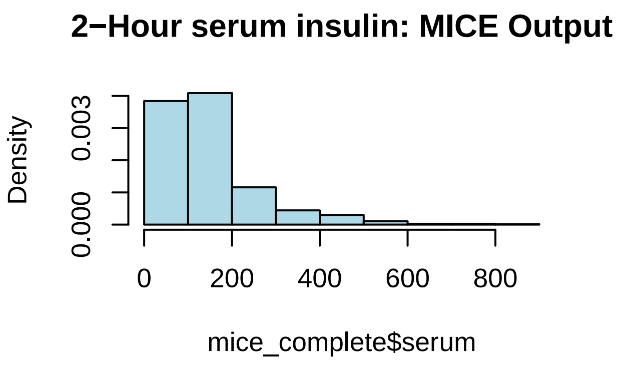
```
##
##  iter imp variable
##   1   1  tsk_thickness  serum
##   1   2  tsk_thickness  serum
##   1   3  tsk_thickness  serum
##   1   4  tsk_thickness  serum
##   1   5  tsk_thickness  serum
##   2   1  tsk_thickness  serum
##   2   2  tsk_thickness  serum
##   2   3  tsk_thickness  serum
##   2   4  tsk_thickness  serum
##   2   5  tsk_thickness  serum
##   3   1  tsk_thickness  serum
##   3   2  tsk_thickness  serum
##   3   3  tsk_thickness  serum
##   3   4  tsk_thickness  serum
##   3   5  tsk_thickness  serum
##   4   1  tsk_thickness  serum
##   4   2  tsk_thickness  serum
##   4   3  tsk_thickness  serum
##   4   4  tsk_thickness  serum
##   4   5  tsk_thickness  serum
##   5   1  tsk_thickness  serum
##   5   2  tsk_thickness  serum
##   5   3  tsk_thickness  serum
##   5   4  tsk_thickness  serum
##   5   5  tsk_thickness  serum
```

```r
# Save the complete imputation output
mice_complete <- complete(mice_mod)

# Show distributions for tsk_thickness and serum
par(mfrow=c(2,2))
hist(dataset$tsk_thickness, freq=F, main='Triceps skin fold thickness : Original Data',
     col='darkgreen', ylim=c(0,0.04))
hist(mice_complete$tsk_thickness, freq=F, main='Triceps skin fold thickness : MICE Output',
     col='lightgreen', ylim=c(0,0.04))
hist(dataset$serum, freq=F, main='2-Hour serum insulin: Original Data',
     col='darkblue', ylim=c(0,0.004))
hist(mice_complete$serum, freq=F, main='2-Hour serum insulin: MICE Output',
     col='lightblue', ylim=c(0,0.004))
```









Compared with the original distributions, the two complete distributions above for tsk_thickness and serum are not significant changed, which are the good things.

```r
# Replace tsk_thickness and serum variables from the mice
dataset$tsk_thickness <- mice_complete$tsk_thickness
dataset$serum <- mice_complete$serum

# Make sure there is no missing data
sum(is.na(dataset))
```

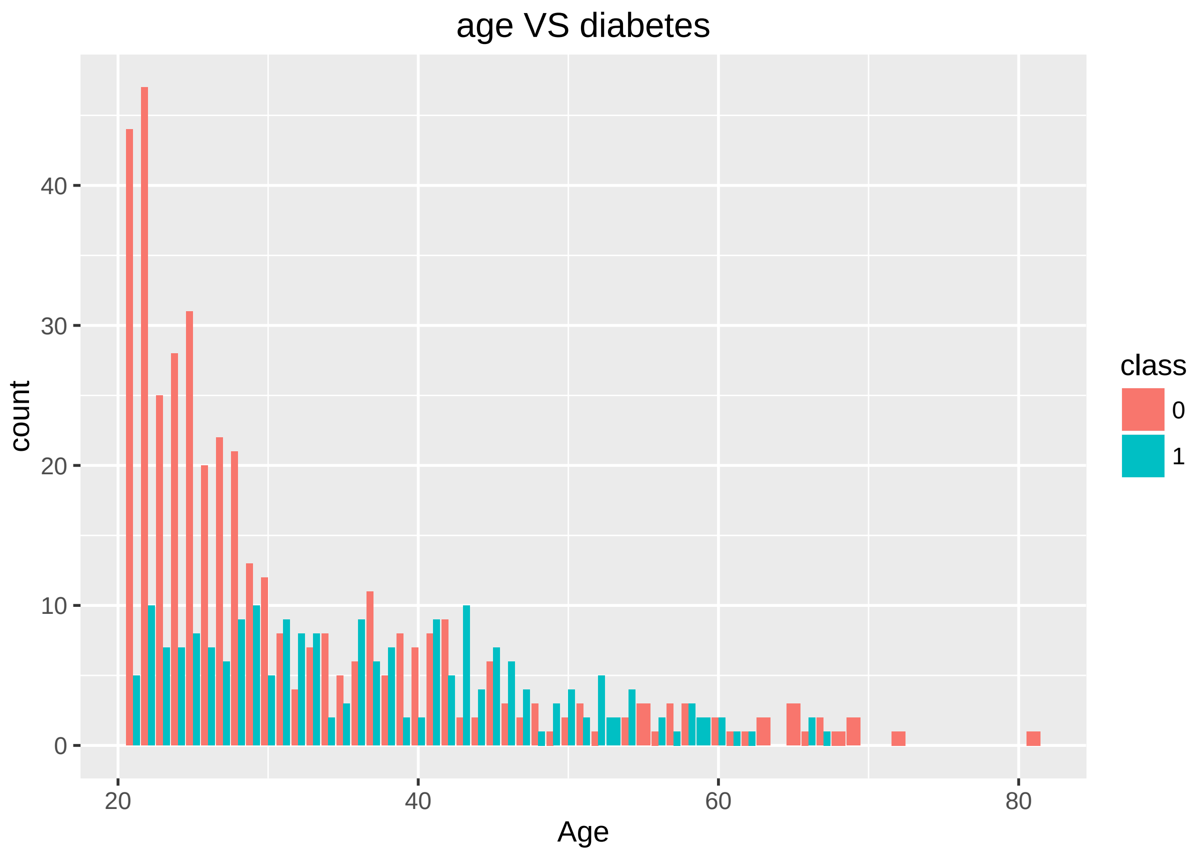
```
## [1] 0
```

Now we have a complete dataset, we can use age and Diabetes pedigree function to do just a bit more feature engineering.

4. Feature Engineering

4.1 Try to Create New Variable

```r
# Visualize the relationship between age and diabetes on training data
 ggplot(data=dataset[train,], aes(x = age, fill = class)) +
   geom_bar(stat='count', position='dodge') +
   ggtitle("age VS diabetes") +
   labs(x = 'Age')
```



Clearly, we can see that thereâ s a age penalty to diabetes on the age large than 30. Initially, I try to collapse this variable into two levels which probably will provide more insights. But, after testing the method on the training data, the result is even worse than before, so right now, I stop here.

4.2 Feature Selection

1. For random forest, it will perform feature selection when we apply the algorithm, because the Gini Impurity method will only choose the variables have significant impact to the result.
2. For logistic regression, we fit a model via penalized maximum likelihood (regularization), which will also do feature selection for us.

5. Build Models

5.1 Normalize the Training Data

```
# Normalize training data
scale_training <- as.data.frame(scale(dataset[train, -9],
                                      center = TRUE, scale = TRUE))

scale_training$class <- dataset[train, "class"]

str(scale_training)
```

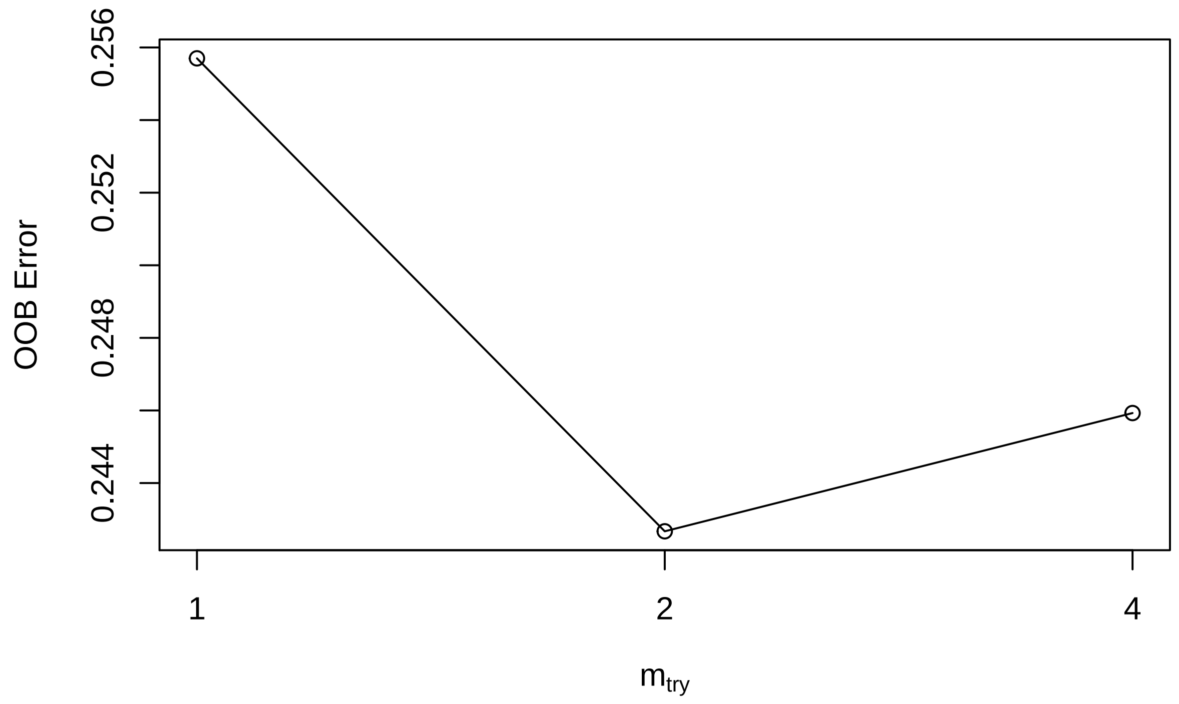
```
## 'data.frame':    614 obs. of  9 variables:
##  $ preg_times   : num  -1.1502 0.0412 -0.2566 0.637 -0.8524 ...
##  $ glucose_test : num  1.851 2.048 -0.252 2.443 -0.416 ...
##  $ blood_press  : num  -1.014 -0.011 -1.85 -0.178 -1.014 ...
##  $ tsk_thickness: num  -0.0405 -0.1352 -1.8392 0.6221 1.5688 ...
##  $ serum        : num  2.784 0.453 -0.571 -0.443 0.223 ...
##  $ bm_index     : num  0.277 -0.598 -0.442 -0.245 0.405 ...
##  $ pedigree_fun : num  1.791 -0.782 0.457 -0.435 -0.175 ...
##  $ age          : num  -1.04 0.248 -0.697 -0.182 -0.783 ...
##  $ class        : Factor w/ 2 levels "0","1": 2 2 1 2 1 1 2 1 2 1 ...
```

5.2 Random Forest

5.21 Find the Optimal Subset for Random Forest

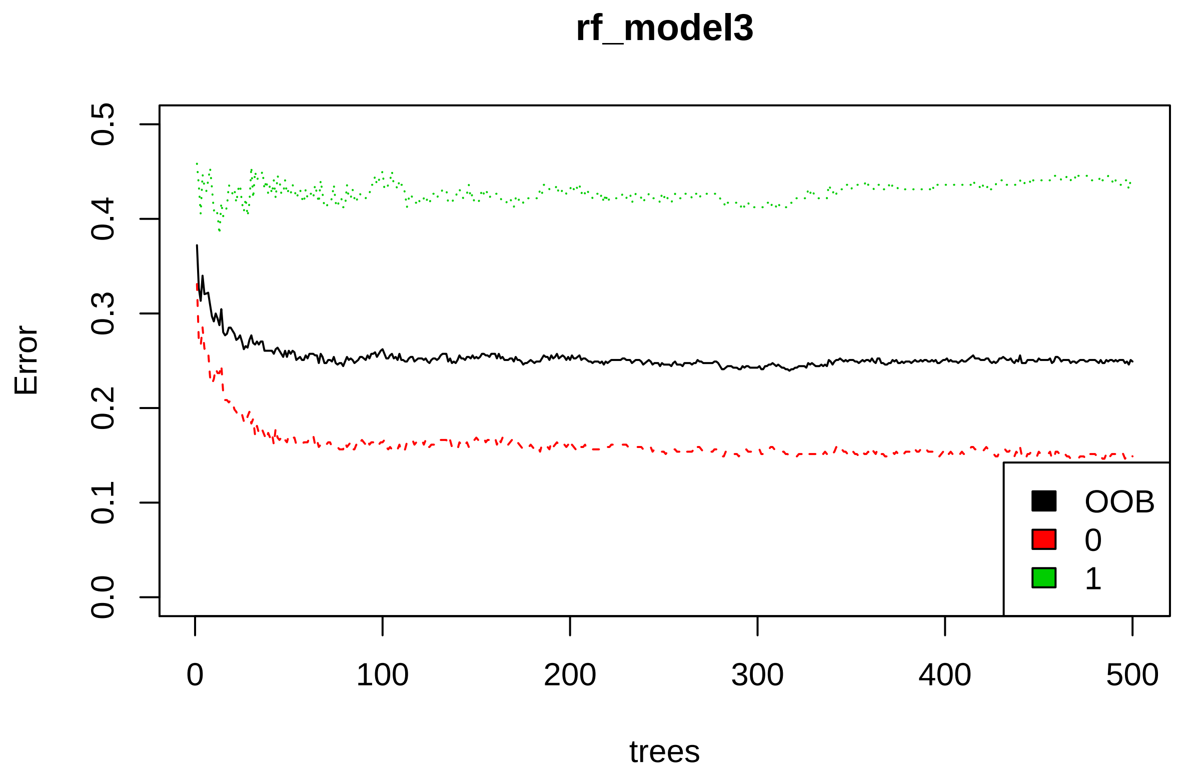
```
bestmtry <- tuneRF(scale_training[, c(-9)],scale_training$class, ntreeTry=300,
                   stepFactor=2,improve=0.05, trace=TRUE, plot=TRUE, dobest=FALSE)
```

```
## mtry = 2  OOB error = 24.27%
## Searching left ...
## mtry = 1     OOB error = 25.57%
## -0.05369128 0.05
## Searching right ...
## mtry = 4     OOB error = 24.59%
## -0.01342282 0.05
```



The result chooses 2 as the optimal number of mtry for each tree. Thus, we use mtry = 2 in the following section.

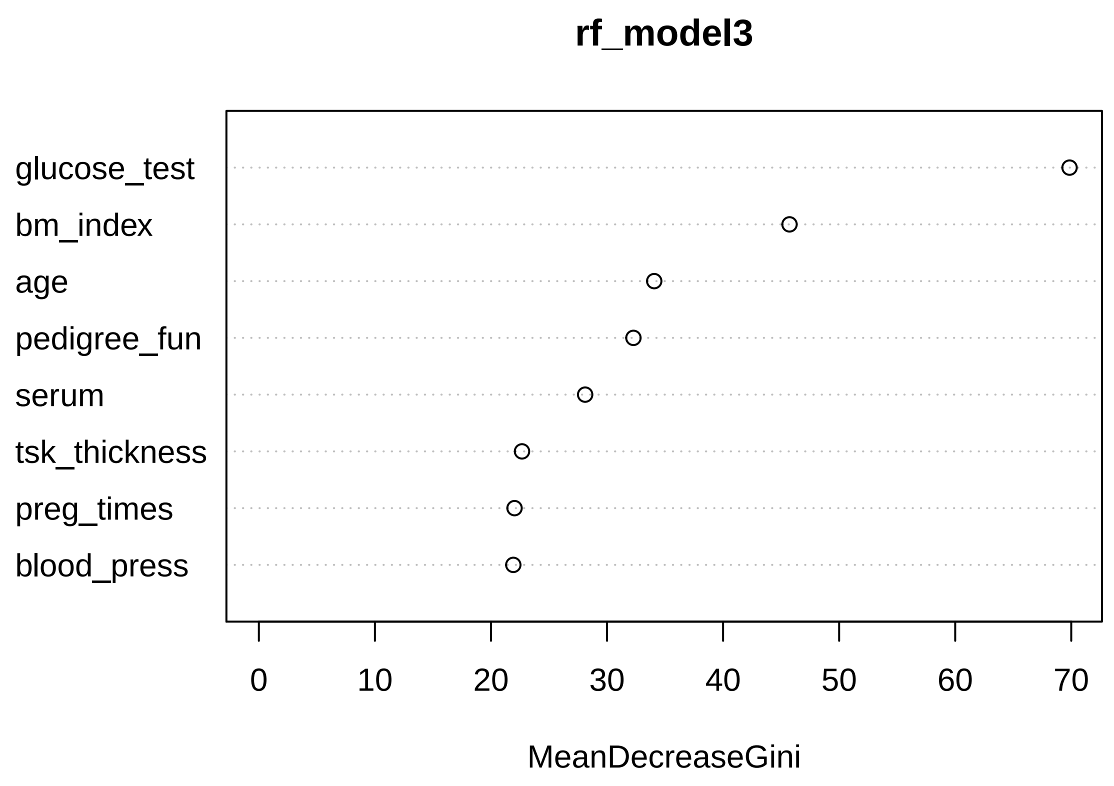
```
rf_model3 <- randomForest(class ~ ., data = scale_training, ntree=500, mtry=2)

# Output classfiaction error rate
plot(rf_model3,ylim = c(0, 0.5))
legend('bottomright', colnames(rf_model3$err.rate), col=1:3, fill=1:3)
```



```
print(rf_model3$err.rate[nrow(rf_model3$err.rate),])
```

```
##       OOB         0         1
## 0.2491857 0.1488834 0.4407583
```

```
# Output variables inportance graph
varImpPlot(rf_model3)
```

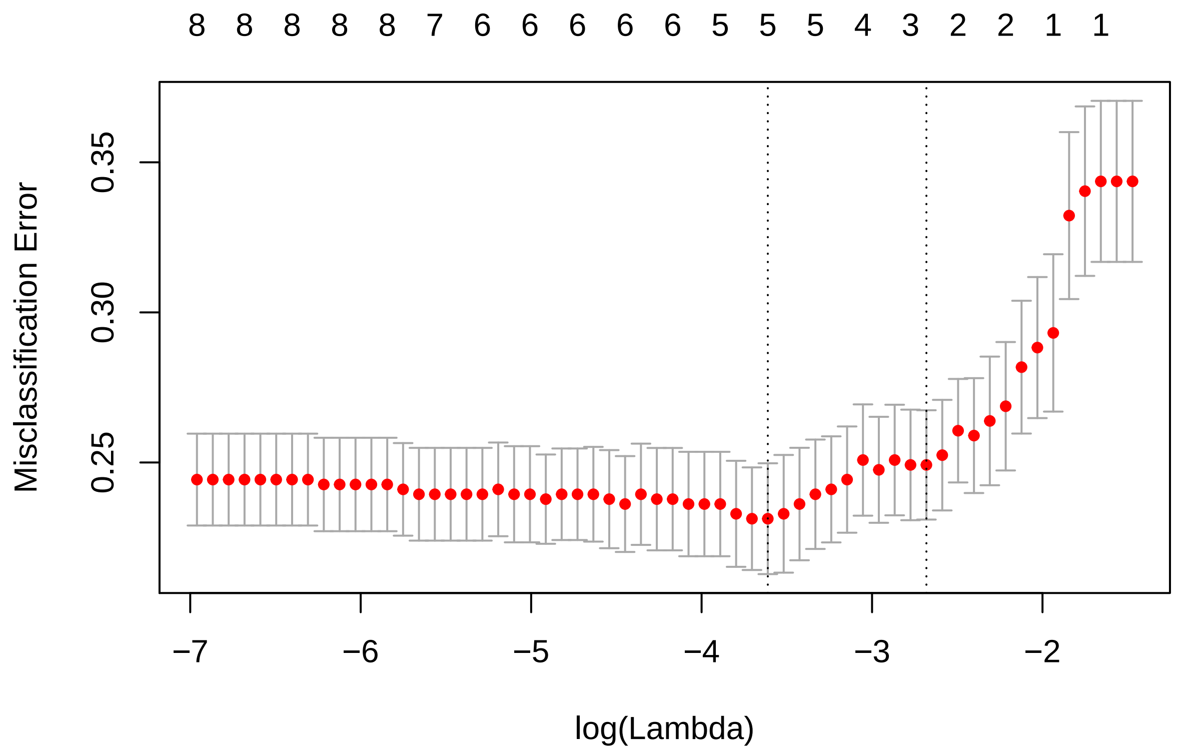


As shown in the Variable Inportance graph above, random forest uses all the features in its algorithm. And the Plasma glucose concentration and Body mass index are sigificant for identifying the diabetes.

5.3 Logistic Regression

Using 10 folds cross-validation to find the best Logistic Regression while avoiding the overfitting by regularization.

```r
cvfit = cv.glmnet(as.matrix(scale_training[, c(-9)]), scale_training$class,
                  family = "binomial", type.measure = "class")

# Show the trend of using different value of the penalized parameter (lambda)
plot(cvfit)
```



```
# Show the cefficients of the best model
coef(cvfit, s = "lambda.min")
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)   -0.79454616
## preg_times     0.23513025
## glucose_test   0.89863688
## blood_press    .
## tsk_thickness  .
## serum          .
## bm_index       0.42152241
## pedigree_fun   0.09236745
## age            0.04708040
```

As shown in the cefficient table above, the best logical regression only uses 5 the features in its algorithm after regularization. Also, it shows Plasma glucose concentration and Body mass index have sigificant impact on identifying the diabetes.

```
lg_p = predict(cvfit, newx = as.matrix(scale_training[, c(-9)]),
            s = "lambda.min", type = "class")

# Show confusion matrix
(lg_result <- table(lg_p, scale_training$class))
```

```
##
## lg_p   0   1
##    0 363 100
##    1  40 111
```

```
# Overall error rate and accuracy
overall_accuracy <- (lg_result[1] +  lg_result[4]) / sum(lg_result)
overall_error <- 1 - overall_accuracy

# Error rate in class 0
error_c0 <- lg_result[2] / (lg_result[1] +  lg_result[2])

# Error rate in class 1
error_c1 <- lg_result[3] / (lg_result[3] +  lg_result[4])
```

6. Comparison and Conclusion

6.1 Model Comparison

Accuracy for logistic regression:

```
overall_accuracy
```

```
## [1] 0.771987
```

Accuracy for random forest:

```
1 - rf_model3$err.rate[nrow(rf_model3$err.rate), 1]
```

```
##       OOB
## 0.7508143
```

The results above suggests that logistic regression performed better than random forest.

6.2 Test Error Rate for Logistic Regression

```
# Normalize the testing dataset
scale_testing <- as.data.frame(scale(dataset[-(train), -9],
                                     center = TRUE, scale = TRUE))
scale_testing$class <- dataset[-(train), "class"]

# Test error for LG
lg_test = predict(cvfit, newx = as.matrix(scale_testing[, c(-9)]),
                  s = "lambda.min", type = "class")

# Show confusion matrix
(lg_result <- table(lg_test, scale_testing$class))
```

```
##
## lg_test  0  1
##       0 88 27
##       1  9 30
```

```r
# Obtain test error and accuracy
test_accuracy <- (lg_result[1] +  lg_result[4]) / sum(lg_result)
test_error <- 1 - overall_accuracy

# Error rate in class 0
(test_c0 <- lg_result[2] / (lg_result[1] +  lg_result[2]))
```

```
## [1] 0.09278351
```

```r
# Error rate in class 1
(test_c1 <- lg_result[3] / (lg_result[3] +  lg_result[4]))
```

```
## [1] 0.4736842
```

The results below show the test error and accuracy for logistic regression.

```r
print(paste("overall test error: ", overall_error))
```

```
## [1] "overall test error:  0.228013029315961"
```

```r
print(paste("overall accuracy: ", overall_accuracy))
```

```
## [1] "overall accuracy:  0.771986970684039"
```

The test error rate is not significant different than the estimate test error, which is obtained from the 10 folds cross-valication. So, we are confident about the logistic regression model is not overfitting.

7. Model Investigattion and Improvements

7.1 Poor Error Rate for Class 1 (Type 2 Error / False Negative Rate)

I find out that both models do a poor job of classifying the patients who have diabetes (class 1). Given this finding, I think we should explore more on it.

The reason for this is that both models are trying to find the lowest total error rate out of all classifier, irrespective of which class the errors come from.

For both random forest and logistic regression, in the two-class case, this result to assign an observation to the class 1 (tested positive for diabetes) if $P(\text{class} = 1 \mid X = x) > 0.5$.

However, for some medical companies might particularly wish to avoid incorrectly classifying an individual who will get diabetes, whereas incorrectly classifying an individual who will not get diabetes, though still to be avoided, is less problematic. I will now see that it is possible to modify logistic regression in order to develop a classifier that better meets this particular requirement.

Right now, we are concerned about incorrectly classifying an individual who will get diabetes, then we can consider lowering this threshold, eg: $P(\text{class} = 1 \mid X = x) > 0.4$. It means we will label any patient with the probability of getting diabetes above 40% to the class 1.

```r
# For training data
lg_train_new <- predict(cvfit, newx = as.matrix(scale_training[, c(-9)]),
                  s = "lambda.min", type = "response")
p_trainclass = ifelse(lg_train_new > 0.4, 1, 0)

# New confusion matrix for training data
new_train <- table(p_trainclass, scale_training$class)

#Error rate in class 0
new_train_c0 <- new_train[2] / (new_train[1] +  new_train[2])

# Error rate in class 1
new_train_c1 <- new_train[3] / (new_train[3] +  new_train[4])

new_overall_accuracy <- (new_train[1] +  new_train[4]) / sum(new_train)
print(paste("new overall training accuracy: ", new_overall_accuracy))
```

```
## [1] "new overall training accuracy:  0.768729641693811"
```

```r
new_overall_error <- 1 - new_overall_accuracy
print(paste("new overall training error: ", new_overall_error))
```

```
## [1] "new overall training error:  0.231270358306189"
```

```r
# For testing data
lg_test_new <- predict(cvfit, newx = as.matrix(scale_testing[, c(-9)]),
                  s = "lambda.min", type = "response")
p_class = ifelse(lg_test_new > 0.4, 1, 0)

# New confusion matrix for testing data
new_decision <- table(p_class, scale_testing$class)

new_overall_accuracy <- (new_decision[1] +  new_decision[4]) / sum(new_decision)
print(paste("new overall test accuracy: ", new_overall_accuracy))
```

```
## [1] "new overall test accuracy:  0.792207792207792"
```

```r
new_overall_error <- 1 - new_overall_accuracy
print(paste("new overall test error: ", new_overall_error))
```

```
## [1] "new overall test error:  0.207792207792208"
```

```r
# Error rate in class 0
new_test_c0 <- new_decision[2] / (new_decision[1] +  new_decision[2])

# Error rate in class 1
new_test_c1 <- new_decision[3] / (new_decision[3] +  new_decision[4])

# Show comparison on training data between old model and new model
(train_matrix <- data.frame(Error_Rate_in_0 = c(error_c0, new_train_c0),
                           Error_Rate_in_1= c(error_c1, new_train_c1),
                           row.names = c("Old Train LG Model", "New Train LG Model")))
```

```
##                     Error_Rate_in_0 Error_Rate_in_1
## Old Train LG Model      0.09925558       0.4739336
## New Train LG Model      0.16377171       0.3601896
```

```
# Show comparison on training data between old model and new model
(test_matrix <- data.frame(Error_Rate_in_0 = c(test_c0, new_test_c0),
                           Error_Rate_in_1= c(test_c1, new_test_c1),
                           row.names = c("Old Test LG Model", "New Test LG Model")))
```

```
##                    Error_Rate_in_0 Error_Rate_in_1
## Old Test LG Model      0.09278351       0.4736842
## New Test LG Model      0.14432990       0.3157895
```

The results in the two data frame above show some interesting points:

- After changing the threshold to 40%, accuracy of the patients who will get diabetes has a vast improvement for both testing and training data.
- By changing the threshold to 40%, the overall accuracy of the model do not chage significantly. The improvement of overall accuracy in testing data is probably because the small size of the testing data.

7.2 Discussion

1. For different purposes of modeling on this dataset may do different experiments on controlling error rate for class 1 (type 2 error). The threshold can be determined and optimized by testing all the probabilities on the training data.
2. It is important to consider the limitations of the given data. This dataset is quite small, which may limit performance of some complicate algorithms.