Predicting Default of Credit Card Holders

Valerii Podymov

June 11, 2017

Overview

The goal of this project is to examine different Machine Learning techniques and compare performance in classification and predictive accuracy among them.

The original data set comes from UCI Machine Learning Repository and also were used in the Kaggle's Default of Credit Card Clients challenge.

Cleaning Data

In order to make data clean we need to provide the right variable types (numeric of factor) and remove missing values and highly correlated variables.

```
#### Load data
raw_data <- read.csv('UCI_Credit_Card.csv')

sum(is.na(raw_data))  ## check missing values
```

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

The data set does not contain missing values.

`PAY` variables have values `-2`, that is not fully documented. Assume this is not an error and means duly payment.

The variables `SEX`, `MARRIAGE` and `EDUCATION` are all factors. Factor levels `5` and `6` in the `EDUCATION` variable have the same meaning `other`, so we can treat them as one factor level.

```
clean_data <- raw_data
clean_data$ID <- NULL

#### sex, education and marriage are factor variables
clean_data$SEX <- as.factor(clean_data$SEX)
clean_data$MARRIAGE <- as.factor(clean_data$MARRIAGE)

#### change the name of the output variable to more convinient
names(clean_data)[length(clean_data)]<-'default'

#### get rid off the extra value 6 in EDUCATION
extra6 <- clean_data$EDUCATION == 6
clean_data[extra6, 'EDUCATION'] <- 5
clean_data$EDUCATION <- as.factor(clean_data$EDUCATION)

#### output variable
clean_data$default <- as.factor(clean_data$default)
```

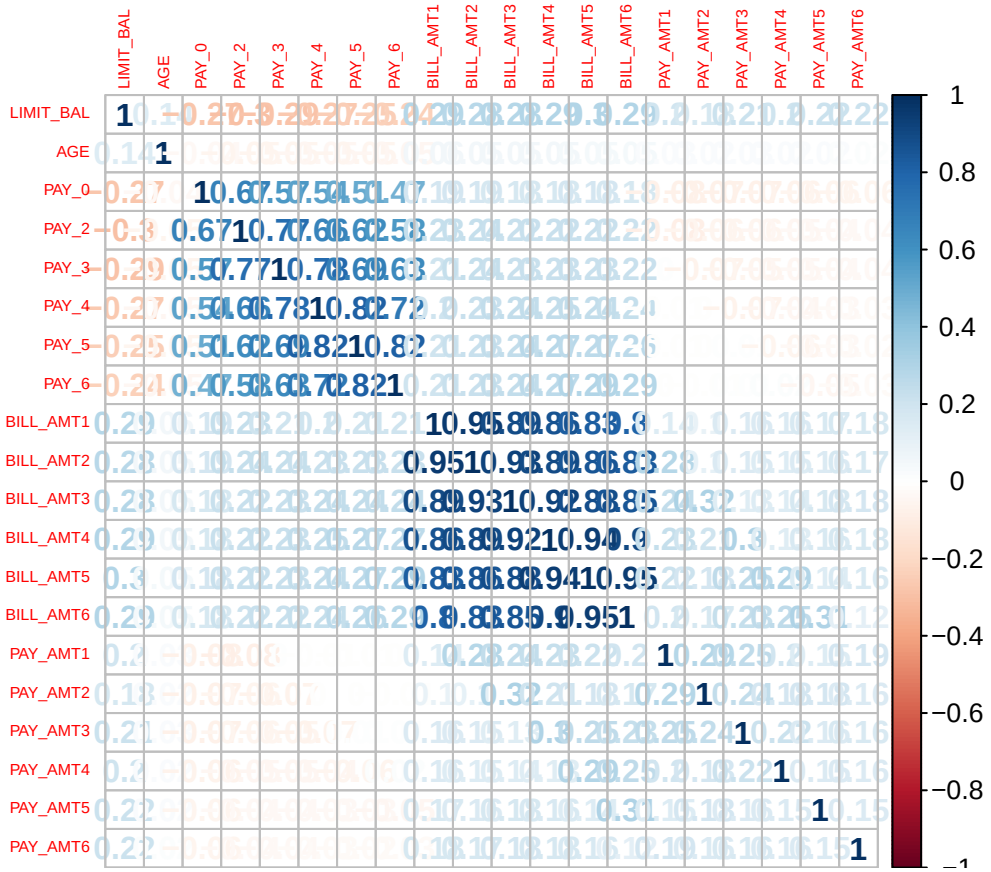
Feature Engineering

The original data set contains 30000 observations of 25 variables. When building a model we want to use only relevant variables as features. The fastest way to do the feature selection is looking at the correlation matrix and excluding highly correlated variables to avoid collinearity in the data. Assume the threshold for high correlation is `0.7`.

```
#### looking at the correlation matrix - numeric vars only
num_cols <- !colnames(clean_data) %in% c('SEX', 'MARRIAGE', 'EDUCATION',
                                         'default')

corr_mtx <- cor(clean_data[, num_cols])
highly_correlated <- findCorrelation(corr_mtx, cutoff = 0.7)
colnames(corr_mtx)[highly_correlated]   ## columns with high correlation
```

```
## [1] "BILL_AMT5" "BILL_AMT4" "BILL_AMT3" "BILL_AMT6" "BILL_AMT2" "PAY_4"
## [7] "PAY_5"     "PAY_3"
```

```
corrplot(corr_mtx, method = 'number', tl.cex = 0.5)
```



The `highly_correlated` array contains the columns highly correlated with others in the data set, and therefore we do not include them in any classifier.

```
# excluding highly correlated variables
clean_data <- clean_data[, -highly_correlated]
```

And the last step in feature engineering is to use one-hot encoding to transform factor variables to a format that works better with classification algorithms.

```
# one-hot-encoding categorical variables
df_dummied <- dummy.data.frame(clean_data,
            names = c('SEX', 'MARRIAGE', 'EDUCATION'), sep="")
```

Splitting Data

We assign 70% of the observations to the training set, and the remaining 30% to the test set. Since the dataset is not particurarly big, and due to the limited time for the task completion, we do not use a validation set to tune the model parameters.

However, **the best practice** is to split the data into three sets to train each model using the training set, then choose the best model using the validation set, and generate the final results using the test set.

```
set.seed(1234)
trainIndices <- createDataPartition(df_dummied$default,
                                    p = 0.7, list = FALSE)
train <- df_dummied[trainIndices, ]
test <- df_dummied[-trainIndices, ]

features <- colnames(train)[1:length(train)-1]
```

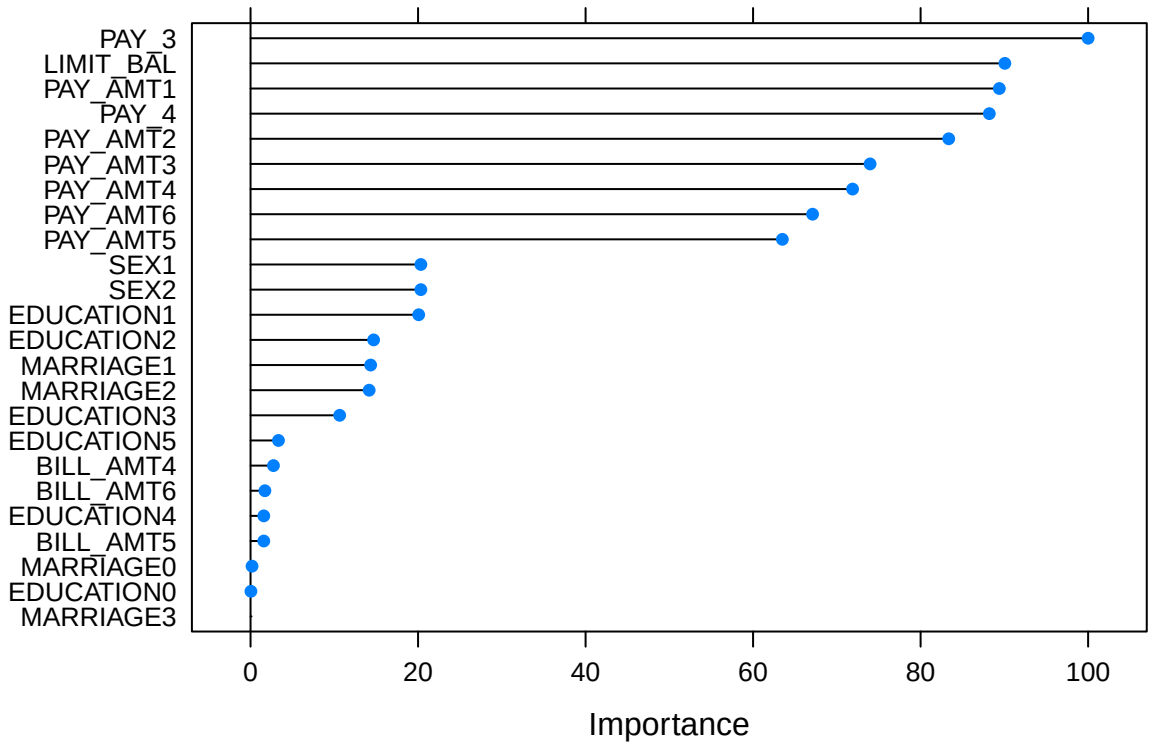
Building Models

We choose cross-validation as a resampling method.

```
# resampling, k-fold cross-validation
fit_ctl <- trainControl(method = 'cv', number = 5)
```

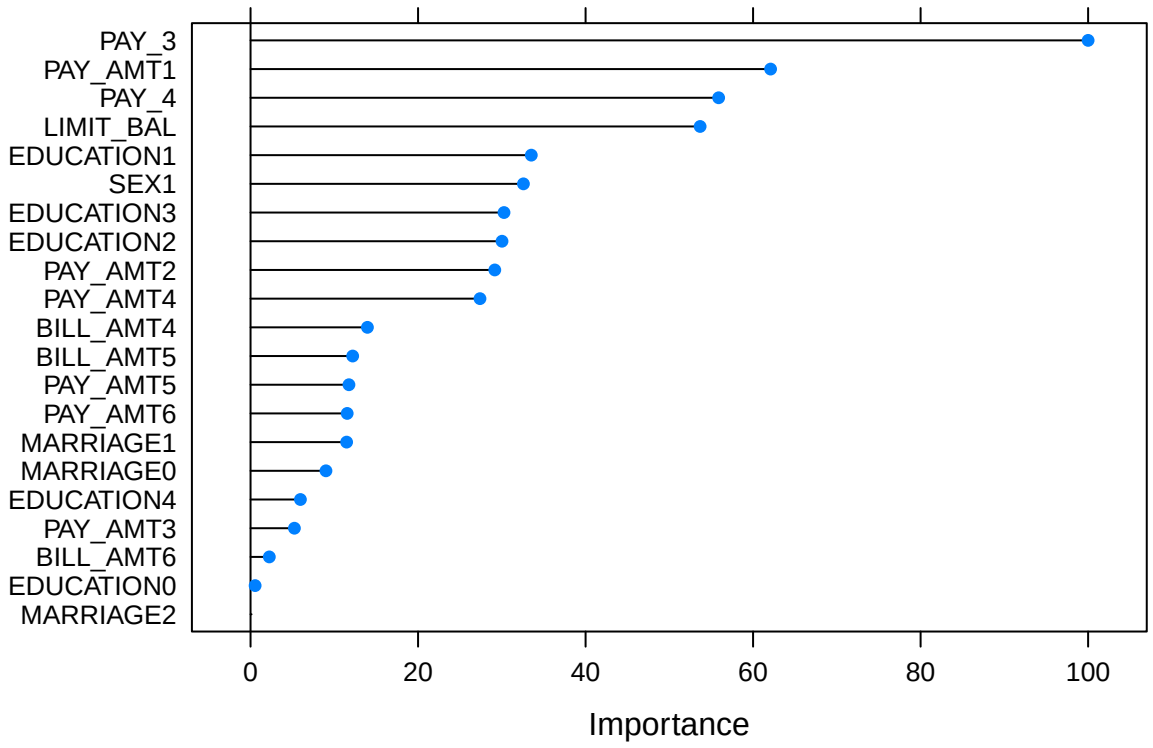
Naive Bayes

```
#### Naive Bayes
fit_nb <- train(x = train[, features], y = train$default,
                method ='nb',
                trControl = fit_ctl)
predict_nb <- predict(fit_nb, newdata = test[, features])
confMat_nb <- confusionMatrix(predict_nb, test$default,
                              positive = '1')
importance_nb <- varImp(fit_nb, scale = TRUE)

plot(importance_nb, main = 'Feature importance for Naive Bayes')
```



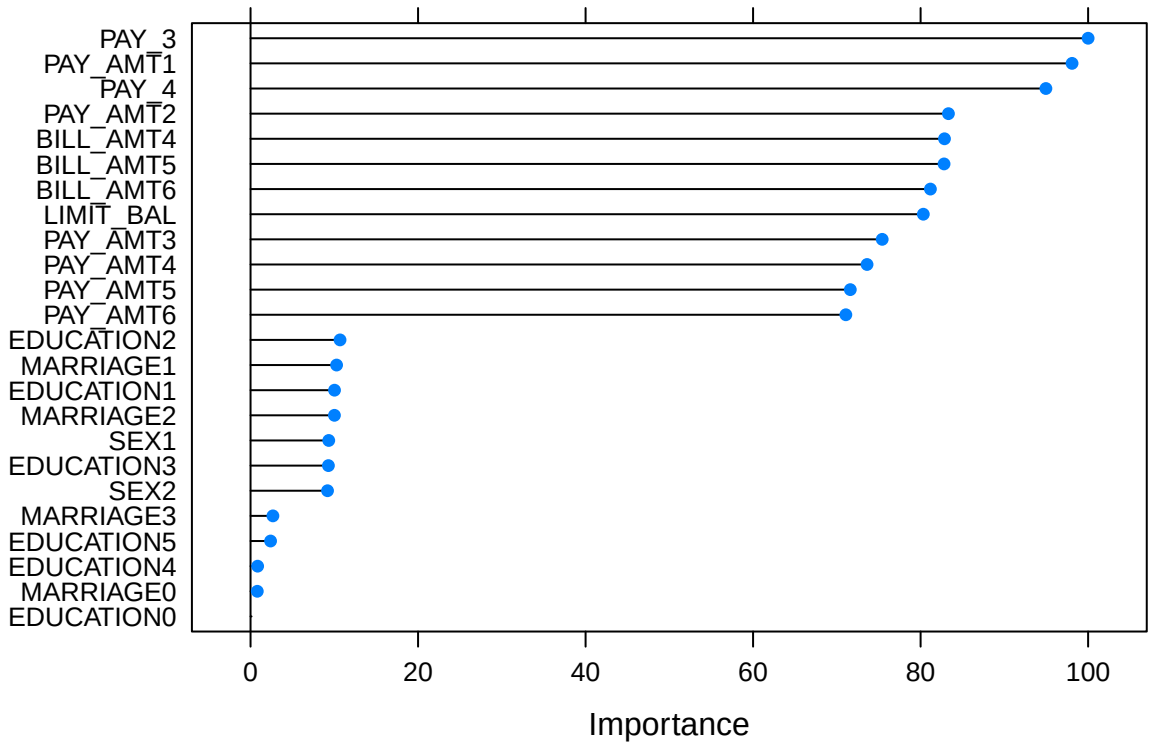
Logistic Regression

```
#### GLM
fit_glm <- train(x = train[, features], y = train$default,
                 method = 'glm',
                 #preProcess = 'range',
                 trControl = fit_ctl)
predict_glm <- predict(fit_glm, newdata = test[, features])
confMat_glm <- confusionMatrix(predict_glm, test$default,
                               positive = '1')
importance_glm <- varImp(fit_glm, scale = TRUE)

plot(importance_glm, main = 'Feature importance for Logistic Regression')
```



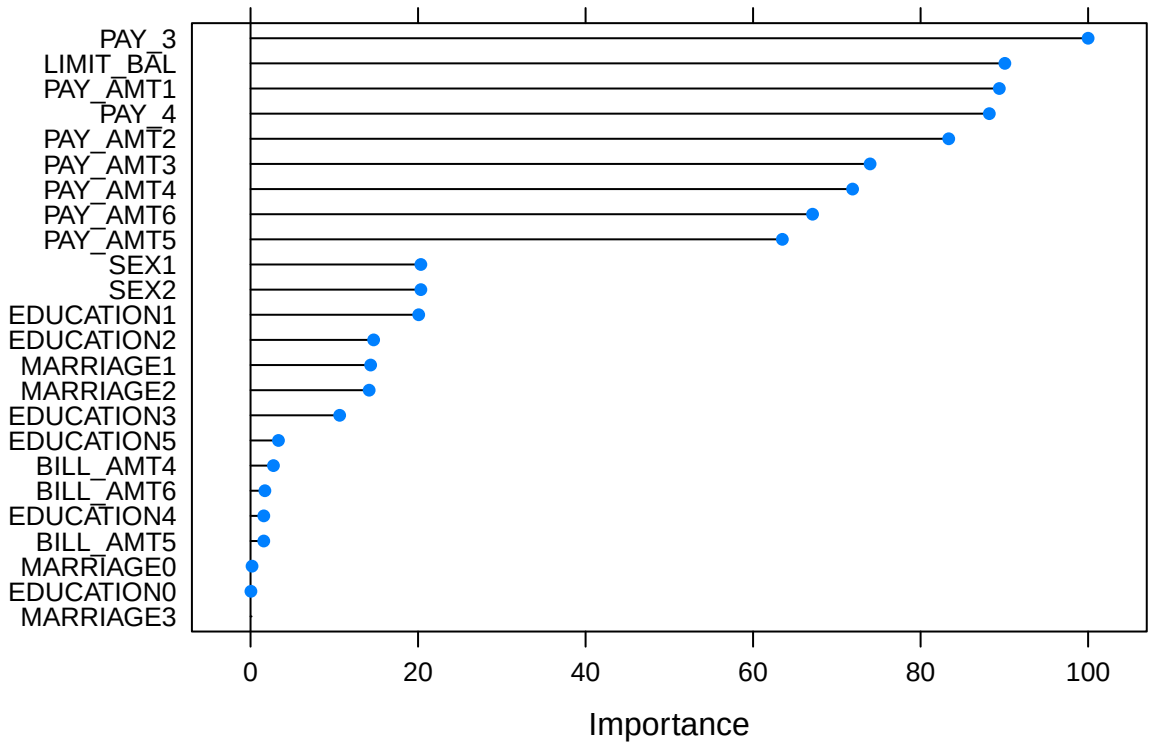
Random Forest

```
#### Random Forest
fit_rf <- train(x = train[, features], y = train$default,
                method = 'rf',
                preProcess = 'range',
                trControl = fit_ctl,
                n.tree = 200)
predict_rf <- predict(fit_rf, newdata = test[, features])
confMat_rf <- confusionMatrix(predict_rf, test$default, positive = '1')
importance_rf <- varImp(fit_rf, scale = TRUE)

plot(importance_rf, main = 'Feature importance for Random Forest')
```



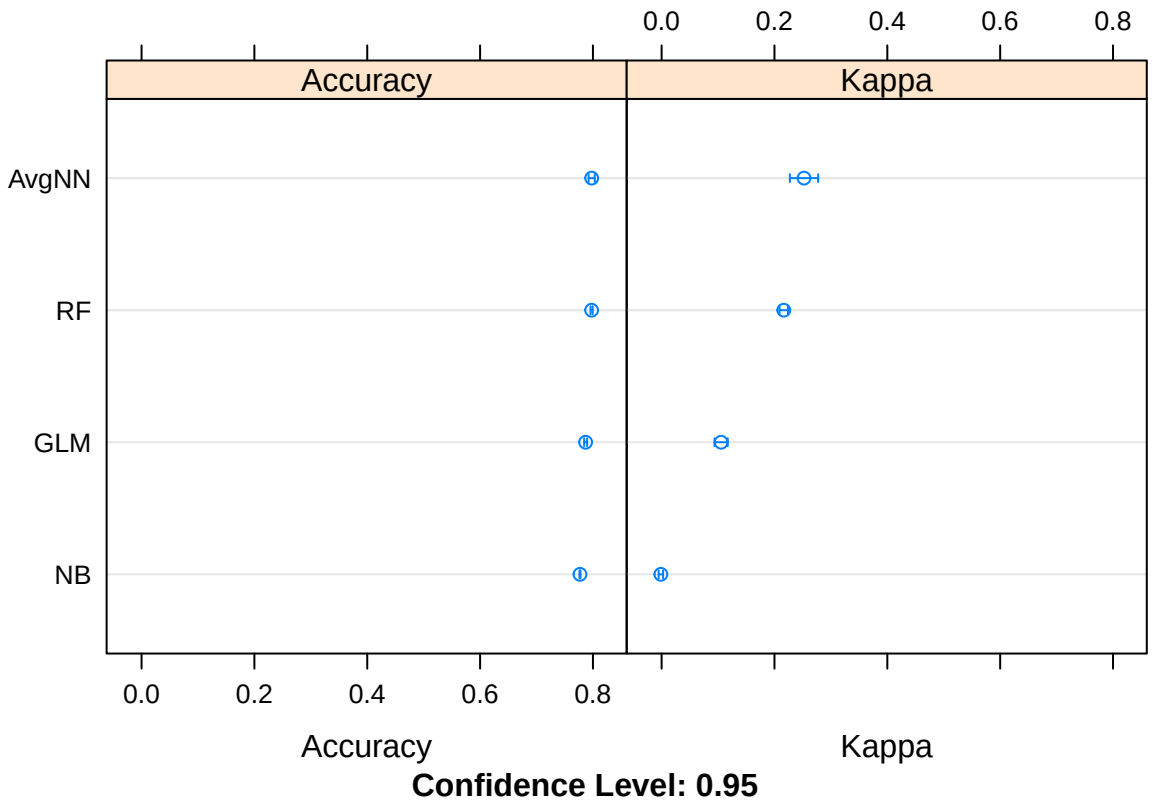
Averaged Neural Networks (ensemble method)

```
# avNNet, ensemble method
fit_nn <- train(x = train[, features], y = train$default,
                    method = 'avNNet',
                    preProcess = 'range',
                    trControl = fit_ctl,
                    tuneGrid = expand.grid(.size = c(3, 5),
                                           .decay = c(0, 0.001, 0.01),
                                           .bag = FALSE),
                    trace = FALSE,
                    maxit = 100)
predict_nn <- predict(fit_nn, newdata = test[, features])
confMat_nn <- confusionMatrix(predict_nn, test$default,
                                  positive = '1')
importance_nn <- varImp(fit_nn, scale = TRUE)

plot(importance_nn, main = 'Feature importance for AvgNN')
```



Comparing the Models

```
models <- resamples(list(NB = fit_nb,
                         GLM = fit_glm,
                         RF = fit_rf,
                         AvgNN = fit_nn
                         ))

dotplot(models)
```



The values of accuracy and Cohen's kappa indicator of concordance show that the Random Forest model (accuracy = 0.7961996; kappa = 0.2170706) and Averaged Neural Networks (accuracy = 0.7966441; kappa = 0.2484808) are both good choice and expected to have a fair out-of-sample error rate.

Additional Notes

- For reproducibility purposes a seed was set in the data splitting phase to obtain same training and testing subsets. However, for full reproducibility we would need to set a seed in the `trainControl` function each time a model is trained. A good example is available on Stackoverflow.
- The other option for feature selection is recursive (forward and backward) feature elimination. For example, with backword elimination we can see how the accuracy of the classifier changes by excluding features from the full feature set. This approach provides accurate results, but it is very slow.
- Different options for resampling such as bootstrapping and repeated k-fold cross validation are also widely discussed, e.g. Bootstrapping vs. Cross-validation
- As it mentioned above, due to the limited time the model parameters are not fine-tuned. Better results would be obtained using Grid Search procedure (which is computationally exhaustive) to find optimal parameters for the models.

Acknowledgements

Lichman, M. (2013). (UCI Machine Learning Repository) [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.