Compulsory Exercise 2: Creditcard fraud detection

Håkon Kjelland-Mørdre Mathias Karsrud Nordal Johan Vik Mathisen

20 April, 2023

Abstract

In this project we investigate the effectiveness of random forest classifiers and support vector machines in classifying fraudulent from non-fraudulent creditcard transactions. The data is PCA transformed transaction data from European cardholders dating back to 2013. The dataset is heavily imbalanced and we have used a combination of SMOTE and naive undersampling to balance it. We found that random forest clasifiers performed well, with over 99.99% sensitivity and a misclassification error of just about 0.15%.

Introduction: Scope and purpose of your project

In this project we aim to find a good classifier for creditcard fraud detection. In particular the task is to classify fraudulent transactions from honest transactions in a heavily imbalanced dataset.

The dataset choosen contains transactions made by European cardholders in September 2013, over a period of two days. It contains only numerical covariates which are the result of a PCA transformation, these are labled `V1-V28`. Additionaly we have `time` and `amount`, which are neither transformes nor scaled and finaly the categorical variable `Class` which denotes whether or not the transaction is fraudulent. There is no further information about the features due to confidentiality issues.

The data is heavily imbalanced, which means that the number of fraudulent transactions is greatly outnumbered by the number of honest transactions. In particular there are 492 frauds out of 284,807 transactions. We attempt to enhance the preformance of our methods by using over- and undesampling techniques. Due to time concerns and computational issues we work on a reduced version of the dataset.

The data is available on Kaggle and has been collected and analysed during a research collaboration of Worldline and the Machine Learning Group of ULB (Université Libre de Bruxelles) on big data mining and fraud detection.

We are trying to find the best performing method within the computational limits of our personal computers. As the covariates are uniformative, we are not particularly interested in which covariates are important predictors, but rather on the performace on the test data.

DATA PRE-PROCESSING

```
set.seed(69)
```

```
# The full dataset:
data <- read.csv("creditcard.csv")

# df.reduced <- read.csv('df.reduced.csv')
```

We begin by scaling the columns `time` and `amount`, such that all features are scaled. If we were not to do this the features `time` and `amount` would play a too large role, because of the large variablility.

```
data$Time <- scale(data$Time)
data$Amount <- scale(data$Amount)
```

In order to deal with in imbalanced dataset there are several possible approaches. We have settled for a hybrid over and undersampling technique. Firstly we use the Synthetic Minority Oversampling Technique (SMOTE) in order to generate synthetic fraudulent transactions close to the original ones in feature space. This is a popular technique providing good results for imbalanced data. We increase the number of fraudulent transactions by a factor of 10. Then we do a naive reduction of the honest transactions by sampling 2% of the orignial data. The undersampling leads to a large information loss and possibility of underfitting the original data. The computational burden of working with the original dataset was too high, which is the reason for the drastic reduction.

After preproccesing the number of fraudulent and non-fraudulent transactions in our dataset is 5412 and 5686 respectively.

```
synthetic_data <- SMOTE(data, data$Class, dup_size = 10)$data
synthetic_data <- synthetic_data %>%
    select(-class)

fraud <- subset(synthetic_data, Class == 1)
non_fraud <- subset(synthetic_data, Class == 0)

reduced_non_fraud <- non_fraud[sample(1:nrow(non_fraud), nrow(non_fraud) * 0.02),
    ]

df.reduced <- rbind(fraud, reduced_non_fraud)[sample(1:(nrow(fraud) + nrow(reduced_non_fraud))),
    ]


print(length(which(df.reduced$Class == 1)))
```

```
## [1] 5412
```

```
print(length(which(df.reduced$Class == 0)))
```

```
## [1] 5686
```

We split our dataset into training and test data by a 70-30 split, with even proportions of fraudulent and honest transactions in both.

```
# even split in each category
even_split <- function(data) {
    frauds <- filter(data, Class == 1)
    non_frauds <- filter(data, Class == 0)

    sample <- sample(c(TRUE, FALSE), nrow(frauds), replace = TRUE, prob = c(0.7,
        0.3))
    even_train_fraud <- data[sample, ]
    even_test_fraud <- data[!sample, ]

    sample <- sample(c(TRUE, FALSE), nrow(non_frauds), replace = TRUE, prob = c(0.7,
        0.3))
    even_train_non_fraud <- data[sample, ]
    even_test_non_fraud <- data[!sample, ]

    even_train <- bind_rows(even_train_fraud, even_train_non_fraud)
    even_test <- bind_rows(even_test_fraud, even_test_non_fraud)

    output <- list(even_train, even_test)
    names(output) <- c("train", "test")
```

```
    return(output)
}

split <- even_split(df.reduced)
df.train <- split$train
df.test <- split$test
```

Desctiprive data analysis/statistics

In order to get a sense of the interaction of the vaiables we visualize the covariance matrix below.

```
ggcorrplot(cor(df.train))
```

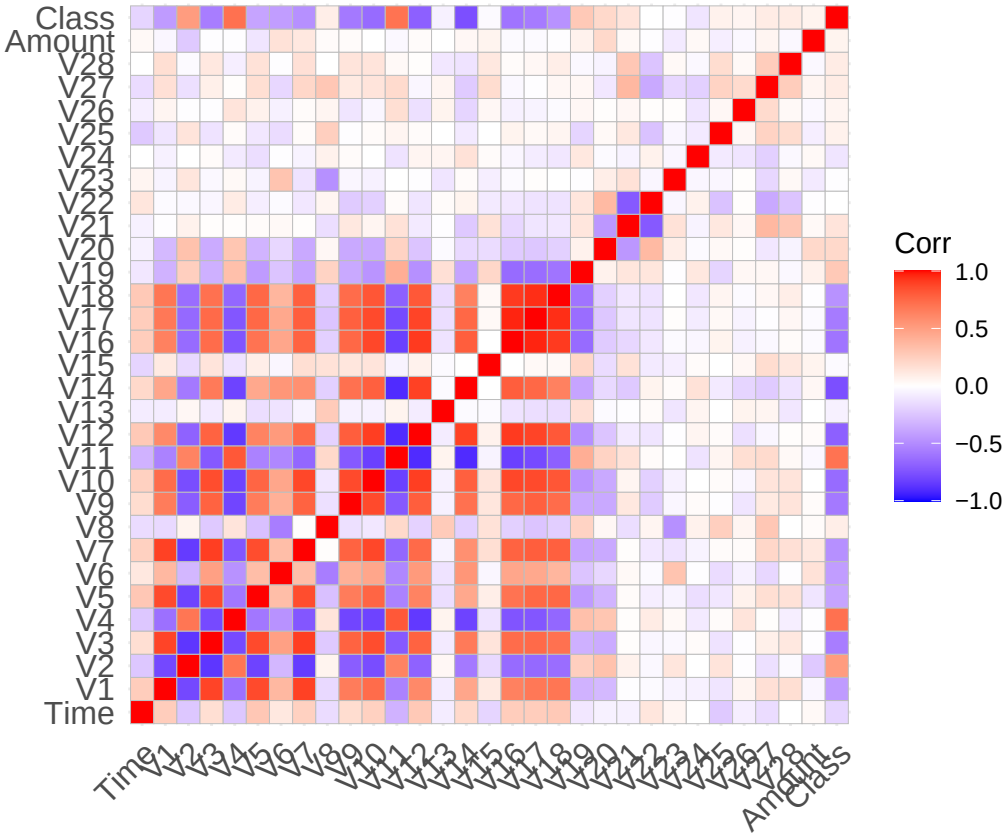

Figure 1: Visualization of the covariance matrix.

For our application the corralations with `Class` is of most interest. We observe that the covariates with the highest correlation values are `V4`, `V11`, `V12` and `V14`. Note also that there is a large correlation between many of the predictors in the feature set.

Below we plot the densities of these covariates for fraudulent and non-fraudulent transactions respectively.

```
# #Transforming regression problem to classification problem
df.train$Class <- factor(df.train$Class, levels = c(0, 1))
df.test$Class <- factor(df.test$Class, levels = c(0, 1))

V4 <- ggplot(data = df.train, mapping = aes(x = V4, y = Class)) + geom_density_ridges(mapping = aes(fil
    bandwidth = 0.181, color = "black", size = 1.5, alpha = 0.8) + xlab("V4") + ggtitle("Fourth principa
```

```
V11 <- ggplot(data = df.train, mapping = aes(x = V11, y = Class)) + geom_density_ridges(mapping = aes(f
    bandwidth = 0.181, color = "black", size = 1.5, alpha = 0.8) + xlab("V11") +
    ggtitle("Eleventh principal component")
V12 <- ggplot(data = df.train, mapping = aes(x = V12, y = Class)) + geom_density_ridges(mapping = aes(f
    bandwidth = 0.181, color = "black", size = 1.5, alpha = 0.8) + xlab("V12") +
    ggtitle("Twelft principal component")
V14 <- ggplot(data = df.train, mapping = aes(x = V14, y = Class)) + geom_density_ridges(mapping = aes(f
    bandwidth = 0.181, color = "black", size = 1.5, alpha = 0.8) + xlab("V14") +
    ggtitle("Fourteenth principal component")

plot_grid(V4, V11, V12, V14, ncol = 2, nrow = 2)
```

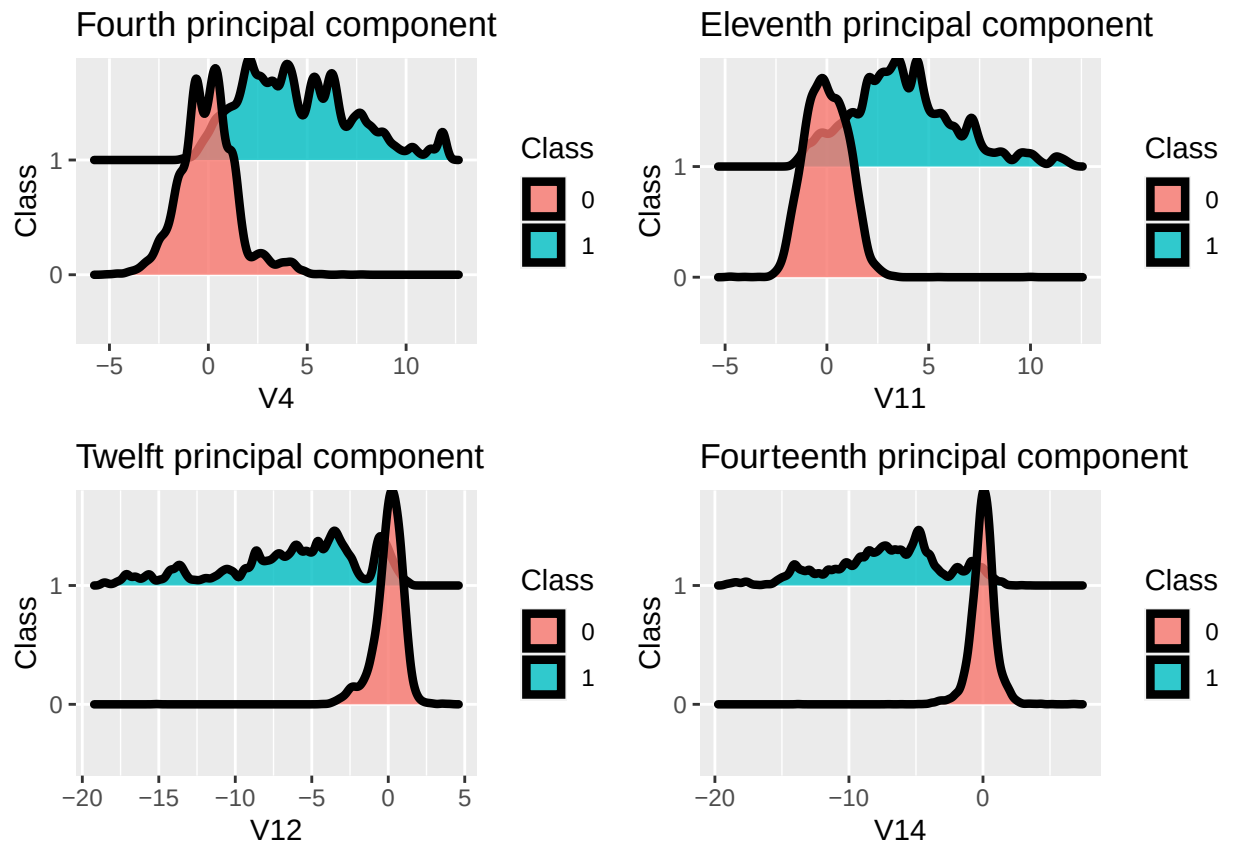

Figure 2: Density plots of aforementioned principal components, separated by fraudulent (1) and non-fraudulent (0).

From the density plots we see that the distributions in Class 1 (fraudulent) are wider, which probably is a consequence of the with oversampling.

Methods

Recall that the data has undergone a PCA transformation, that is all of the features are principle components with no real interpretation. Hence, a models interpretability has less to no effect on the choice of models used in this project. We have chosen to investigate two methods, random forest and support vector machines.

First, we implement some functions that will be used in assessing the models.

```
# Misclassification rate
missclass <- function(model, preds, testRespons) {
    mc <- table(preds, testRespons)
    return(1 - sum(diag(mc))/sum(mc))
}

# Precision
prec <- function(model, pred, testRespons) {
    mc <- table(pred, testRespons)
    return(mc[2, 2]/(sum(mc[2, ])))
}
# Sensitivity
sens <- function(model, pred, testRespons) {
    mc <- table(pred, testRespons)
    return(mc[2, 2]/(sum(mc[, 2])))
}
```

Random Forest Random forest usually perform well on large datasets with many features, due to its ability to identify complex patterns and interactions between the many features. From the correlation matrix shown earlier, we observed a large amount of correlation between many of the variables. There are probably many different factors that may be indicative of fraudulent behavior, and these factors may interact in non-linear ways. Random forests are particularly good at capturing these types of complex relationships.

To do so the random forest algorithm combines multiple decision trees to make more accurate predictions. When growing each tree, the algorithm randomly selects a subset of the predictors to be used for the split at each node. Finally, the method aggregates the results of all the trees to make a final prediction. The randomization helps to reduce overfitting and improve the generalization performance of the model.

```
# Function returning random forest model
RF <- function(d, m, n) {
    rf.mod <- randomForest(formula = Class ~ ., mtry = m, data = d, importance = T,
        ntree = n)
    return(rf.mod)
}
```

Let $p = 31$ be the number of predictors, and define M and N as follows:

$M = 10, \sqrt{p}, p$ is the set of values for the number of predictors chosen when performing each split.

$N = 50, 100, 150$ is the set of values for the number of trees used in the model.

In other words, M represents the set of values for the hyperparameter "mtry", which determines the number of predictors randomly sampled at each split, while N represents the set of values for the hyperparameter "ntree", which determines the number of decision trees in the random forest model.

Note, that choosing p number of predictors when performing each split is equivalent to the bagging procedure.

We will use the values presented above to compare nine models, and obtain the optimal model with respect to these parameters.

```
rf.mod1.sqrt <- RF(df.train, sqrt(ncol(df.train)), 50)
rf.mod2.sqrt <- RF(df.train, sqrt(ncol(df.train)), 100)
rf.mod3.sqrt <- RF(df.train, sqrt(ncol(df.train)), 150)
```

```
rf.mod1.bag <- RF(df.train, ncol(df.train), 50)
rf.mod2.bag <- RF(df.train, ncol(df.train), 100)
rf.mod3.bag <- RF(df.train, ncol(df.train), 150)
```

```
rf.mod1.ten <- RF(df.train, 10, 50)
rf.mod2.ten <- RF(df.train, 10, 100)
rf.mod3.ten <- RF(df.train, 10, 150)
```

We now want use our nine models to make predictions on our test data

```
# mtry = sqrt(p)
rf.pred1.sqrt <- predict(rf.mod1.sqrt, newdata = df.test)
rf.pred2.sqrt <- predict(rf.mod2.sqrt, newdata = df.test)
rf.pred3.sqrt <- predict(rf.mod3.sqrt, newdata = df.test)

# mtry = p
rf.pred1.bag <- predict(rf.mod1.bag, newdata = df.test)
rf.pred2.bag <- predict(rf.mod2.bag, newdata = df.test)
rf.pred3.bag <- predict(rf.mod3.bag, newdata = df.test)

# mtry = 10
rf.pred1.ten <- predict(rf.mod1.ten, newdata = df.test)
rf.pred2.ten <- predict(rf.mod2.ten, newdata = df.test)
rf.pred3.ten <- predict(rf.mod3.ten, newdata = df.test)
```

```
# mtry = sqrt(p), trees = 50
missclass1.sqrt <- missclass(rf.mod1.sqrt, rf.pred1.sqrt, df.test$Class)
sens1.sqrt <- sens(rf.mod1.sqrt, rf.pred1.sqrt, df.test$Class)
prec1.sqrt <- prec(rf.mod1.sqrt, rf.pred1.sqrt, df.test$Class)


# mtry = sqrt(p), trees = 100
missclass2.sqrt <- missclass(rf.mod2.sqrt, rf.pred2.sqrt, df.test$Class)
sens2.sqrt <- sens(rf.mod2.sqrt, rf.pred2.sqrt, df.test$Class)
prec2.sqrt <- prec(rf.mod2.sqrt, rf.pred2.sqrt, df.test$Class)


# mtry = sqrt(p), trees = 150
missclass3.sqrt <- missclass(rf.mod3.sqrt, rf.pred3.sqrt, df.test$Class)
sens3.sqrt <- sens(rf.mod3.sqrt, rf.pred3.sqrt, df.test$Class)
prec3.sqrt <- prec(rf.mod3.sqrt, rf.pred3.sqrt, df.test$Class)


# mtry = p, trees = 50
missclass1.bag <- missclass(rf.mod1.bag, rf.pred1.bag, df.test$Class)
sens1.bag <- sens(rf.mod1.bag, rf.pred1.bag, df.test$Class)
prec1.bag <- prec(rf.mod1.bag, rf.pred1.bag, df.test$Class)

# mtry = p, trees = 100
missclass2.bag <- missclass(rf.mod2.bag, rf.pred2.bag, df.test$Class)
sens2.bag <- sens(rf.mod2.bag, rf.pred2.bag, df.test$Class)
prec2.bag <- prec(rf.mod2.bag, rf.pred2.bag, df.test$Class)


# mtry = p, trees = 150
missclass3.bag <- missclass(rf.mod3.bag, rf.pred3.bag, df.test$Class)
sens3.bag <- sens(rf.mod3.bag, rf.pred3.bag, df.test$Class)
prec3.bag <- prec(rf.mod3.bag, rf.pred3.bag, df.test$Class)
```

```r
# mtry = 10, trees = 50
missclass1.ten <- missclass(rf.mod1.ten, rf.pred1.ten, df.test$Class)
sens1.ten <- sens(rf.mod1.ten, rf.pred1.ten, df.test$Class)
prec1.ten <- prec(rf.mod1.ten, rf.pred1.ten, df.test$Class)


# mtry = 10, trees = 100
missclass2.ten <- missclass(rf.mod2.ten, rf.pred2.ten, df.test$Class)
sens2.ten <- sens(rf.mod2.ten, rf.pred2.ten, df.test$Class)
prec2.ten <- prec(rf.mod2.ten, rf.pred2.ten, df.test$Class)


# mtry = 10, trees = 150
missclass3.ten <- missclass(rf.mod3.ten, rf.pred3.ten, df.test$Class)
sens3.ten <- sens(rf.mod3.ten, rf.pred3.ten, df.test$Class)
prec3.ten <- prec(rf.mod3.ten, rf.pred3.ten, df.test$Class)
```

```r
# mtry, trees, Missclass. rate, Sens
rf1 <- c("sqrt(p)", 50, round(missclass1.sqrt, 6), round(sens1.sqrt, 6), round(prec1.sqrt,
    6))

rf2 <- c("sqrt(p)", 100, round(missclass2.sqrt, 6), round(sens2.sqrt, 6), round(prec2.sqrt,
    6))

rf3 <- c("sqrt(p)", 150, round(missclass3.sqrt, 6), round(sens3.sqrt, 6), round(prec3.sqrt,
    6))

###

rf4 <- c("p", 50, round(missclass1.bag, 6), round(sens1.bag, 6), round(prec1.bag,
    6))

rf5 <- c("p", 100, round(missclass2.bag, 6), round(sens2.bag, 6), round(prec2.bag,
    6))

rf6 <- c("p", 150, round(missclass3.bag, 6), round(sens3.bag, 6), round(prec3.bag,
    6))

###

rf7 <- c(10, 50, round(missclass1.ten, 6), round(sens1.ten, 6), round(prec1.ten,
    6))

rf8 <- c(10, 100, round(missclass2.ten, 6), round(sens2.ten, 6), round(prec2.ten,
    6))

rf9 <- c(10, 150, round(missclass3.ten, 6), round(sens3.ten, 6), round(prec3.ten,
    6))


result_table <- rbind(rf1, rf2, rf3, rf4, rf5, rf6, rf7, rf8, rf9)
```

SVM The support vector machine, SVM, algorithm is attractive mostly for the same reasons as random forest; SVM perform well on large datasets with large associated feature spaces, as well as being good at capturing complex, non-linear relations between the covariates.

The SVM tries to find the boundary separating the two classes with the largest margin possible. Margin being the distance between the boundary and the closest sample point from each class. The algorithm transforms the input data into a higher dimensional space through what is known as a kernel trick. In the higher dimensional space the algorithm finds the optimal hyperplane separating the two classes. The decision boundary in the original feature space is then the projection of this hyperplane onto the original space.

It is clear that the kernel is a very important hyperparameter of the SVM. There are many different kernels that could be used, however the goal of selecting a kernel function is to transform the input data into a higher-dimensional space where it can be more easily separated by a hyperplane. Thus comparing different kernels is important when using SVM.

Another important parameter of the SVM is the cost parameter. In short it provides a penalty for misclassifying a data point in the training set. That is, a higher value of the cost parameter will result in smaller margin and a more complex boundary, whilst a smaller cost value yields a larger margin and a less complex boundary. The choice of cost parameter is therefore a suspect to the bias-variacne trade-off.

Let $C = \{0.1, 1, 10\}$ be the set of cost values we will investigate. We will create three models for each of the three kernels linear, polynomial and radial, corresponding to the elements of C.

```
# Radial kernel
svm.rad.01 <- svm(Class ~ ., df.train, kernel = "radial", cost = 0.1)
svm.rad.1 <- svm(Class ~ ., df.train, kernel = "radial", cost = 1)
svm.rad.10 <- svm(Class ~ ., df.train, kernel = "radial", cost = 10)

# Linear kernel
svm.lin.01 <- svm(Class ~ ., df.train, kernel = "linear", cost = 0.1)
svm.lin.1 <- svm(Class ~ ., df.train, kernel = "linear", cost = 1)
svm.lin.10 <- svm(Class ~ ., df.train, kernel = "linear", cost = 10)

# Polynomial kernel
svm.poly.01 <- svm(Class ~ ., df.train, kernel = "polynomial", cost = 0.1)
svm.poly.1 <- svm(Class ~ ., df.train, kernel = "polynomial", cost = 1)
svm.poly.10 <- svm(Class ~ ., df.train, kernel = "polynomial", cost = 10)
```

We now want use our SVM-models to make predictions on our test set.

```
# rad
pred.rad.01 <- predict(svm.rad.01, newdata = df.test)
pred.rad.1 <- predict(svm.rad.1, newdata = df.test)
pred.rad.10 <- predict(svm.rad.10, newdata = df.test)

# lin
pred.lin.01 <- predict(svm.lin.01, newdata = df.test)
pred.lin.1 <- predict(svm.lin.1, newdata = df.test)
pred.lin.10 <- predict(svm.lin.10, newdata = df.test)

# poly
pred.poly.01 <- predict(svm.poly.01, newdata = df.test)
pred.poly.1 <- predict(svm.poly.1, newdata = df.test)
pred.poly.10 <- predict(svm.poly.10, newdata = df.test)
```

```
# rad, c = 0.1
misclass01.rad <- missclass(svm.rad.01, pred.rad.01, df.test$Class)
sens01.rad <- sens(svm.rad.01, pred.rad.01, df.test$Class)
```

```r
prec01.rad <- prec(svm.rad.01, pred.rad.01, df.test$Class)

# rad, c = 1
misclass1.rad <- missclass(svm.rad.1, pred.rad.1, df.test$Class)
sens1.rad <- sens(svm.rad.1, pred.rad.1, df.test$Class)
prec1.rad <- prec(svm.rad.1, pred.rad.1, df.test$Class)

# rad, c = 10
misclass10.rad <- missclass(svm.rad.10, pred.rad.10, df.test$Class)
sens10.rad <- sens(svm.rad.10, pred.rad.10, df.test$Class)
prec10.rad <- prec(svm.rad.10, pred.rad.10, df.test$Class)


# lin, c = 0.1
misclass01.lin <- missclass(svm.lin.01, pred.lin.01, df.test$Class)
sens01.lin <- sens(svm.lin.01, pred.lin.01, df.test$Class)
prec01.lin <- prec(svm.lin.01, pred.lin.01, df.test$Class)


# lin, c = 1
misclass1.lin <- missclass(svm.lin.1, pred.lin.1, df.test$Class)
sens1.lin <- sens(svm.lin.1, pred.lin.1, df.test$Class)
prec1.lin <- prec(svm.lin.1, pred.lin.1, df.test$Class)


# lin, c = 10
misclass10.lin <- missclass(svm.lin.10, pred.lin.10, df.test$Class)
sens10.lin <- sens(svm.lin.10, pred.lin.10, df.test$Class)
prec10.lin <- prec(svm.lin.10, pred.lin.10, df.test$Class)


# poly, c = 0.1
misclass01.poly <- missclass(svm.poly.01, pred.poly.01, df.test$Class)
sens01.poly <- sens(svm.poly.01, pred.poly.01, df.test$Class)
prec01.poly <- prec(svm.poly.01, pred.poly.01, df.test$Class)


# poly, c = 1
misclass1.poly <- missclass(svm.poly.1, pred.poly.1, df.test$Class)
sens1.poly <- sens(svm.poly.1, pred.poly.1, df.test$Class)
prec1.poly <- prec(svm.poly.1, pred.poly.1, df.test$Class)


# poly, c = 10
misclass10.poly <- missclass(svm.poly.10, pred.poly.10, df.test$Class)
sens10.poly <- sens(svm.poly.10, pred.poly.10, df.test$Class)
prec10.poly <- prec(svm.poly.10, pred.poly.10, df.test$Class)
```

```r
svm1 <- c(0.1, "Rad", misclass01.rad, sens01.rad, prec01.rad)

svm2 <- c(1, "Rad", misclass1.rad, sens1.rad, prec1.rad)

svm3 <- c(10, "Rad", misclass10.rad, sens10.rad, prec10.rad)
```

```
###

svm4 <- c(0.1, "Lin", misclass01.lin, sens01.lin, prec01.lin)

svm5 <- c(1, "Lin", misclass1.lin, sens1.lin, prec1.lin)

svm6 <- c(10, "Lin", misclass10.lin, sens10.lin, prec10.lin)

###

svm7 <- c(0.1, "Poly", misclass01.poly, sens01.poly, prec01.poly)

svm8 <- c(1, "Poly", misclass1.poly, sens1.poly, prec1.poly)

svm9 <- c(10, "Poly", misclass10.poly, sens10.poly, prec10.poly)

result_table_svm <- rbind(svm1, svm2, svm3, svm4, svm5, svm6, svm7, svm8, svm9)
```

Evaluation metrics We will compare the performance of the models, with respect to their precision, misclassification rate and their sensitivity. Recall that precision measures the proportion of true positives among all instances that are predicted as positive. The misclassification rate is the proportion of misclasified instances among all instances in the dataset and that sensitivity is defined as the proportion of true positive cases that are correctly classified by the model. In this case sensitivity will be the ratio of frauds classified as frauds by the model and the total number of frauds in the data. Since falsely classifying a fraud as a non-fraud can have serious consequences, such as financial losses or damage to a company's reputation, a high sensitivity is particularly important for a fraud detection model.

Hyperparameters As previously discussed, there are nine models created by varying two different parameters for each of the methods used. For the random forest the models where made while varying the number of trees and the number of predictors used for each split. For the SVM, the kernel and the cost parameter was changed for the different models. Finding the optimal parameters for each method will be done through examining the metrics described above.

It should be noted that, ideally, a cross-validation should have been performed to obtain the best cost parameters for each kernel in the SVM model. However, this was not feasible due to its high computational demand. Additionally, for the radial kernel, there is a parameter called γ which we did not optimize in our project and left at its default value. Ideally, a cross-validation should also be conducted to determine the optimal value for this parameter, but again, it was too computationally expensive to do so. Nevertheless, it is worth noting that the default value of γ is selected appropriately to fit the data, and therefore, it is assumed that this value is not far from the optimal value.

Results and interpretation

We present the various evaluation metrics based on the predictions for all models.

```
# Random forest results

knitr::kable(result_table, "latex", booktabs = TRUE, escape = FALSE, caption = "Model evaluations for Ra
    col.names = c("Mtry", "Trees", "Missclassification rate", "Sensitivity", "Precision")) %>%
    kable_styling(latex_options = "HOLD_position")
```

Table 1: Model evaluations for Random forrest

	Mtry	Trees	Missclassification rate	Sensitivity	Precision
rf1	sqrt(p)	50	0.002448	0.994956	1
rf2	sqrt(p)	100	0.001836	0.996217	1
rf3	sqrt(p)	150	0.002142	0.995586	1
rf4	p	50	0.003671	0.993695	0.998733
rf5	p	100	0.003977	0.993695	0.9981
rf6	p	150	0.003977	0.993695	0.9981
rf7	10	50	0.002142	0.996217	0.999367
rf8	10	100	0.002448	0.994956	1
rf9	10	150	0.002142	0.995586	1

We observe that the random forest method overall perform well, with very minor differnces along all metrics. The model with 150 trees and Mtry = $\sqrt{p}$ (5.576) scores best in all three. It is worth noticing is that the models with lower Mtry perform more evenly. There are many covariates and the full models are in risk of overfitting the training data. By evaluating fewer covariates in each step we achieve better generalization to the test data. Why fewer number of trees results in the best model, seems to be a coincident. We conclude that the parameter of importance is the number of predictors chosen at each split.

```
# SVM results
knitr::kable(result_table_svm, "latex", booktabs = TRUE, escape = FALSE, caption = "Model evaluations f
    col.names = c("c", "Kernel", "Missclassification rate", "Sensitivity", "Precision")) %>%
    kable_styling(latex_options = "HOLD_position")
```

Table 2: Model evaluations for SVM

	c	Kernel	Missclassification rate	Sensitivity	Precision
svm1	0.1	Rad	0.0494110448217837	0.905107187894073	0.992395437262357
svm2	1	Rad	0.0276885421447147	0.953341740226986	0.989205103042198
svm3	10	Rad	0.00979042374177752	0.987704918032787	0.992083597213426
svm4	0.1	Lin	0.0566008872571516	0.901324085750315	0.980452674897119
svm5	1	Lin	0.0547651828055683	0.907313997477932	0.978246091094494
svm6	10	Lin	0.0538473305797766	0.909520807061791	0.977966101694915
svm7	0.1	Poly	0.0592014685635612	0.880201765447667	0.997499106823866
svm8	1	Poly	0.0361021875478048	0.932219419924338	0.992948287441236
svm9	10	Poly	0.0125439804191525	0.98266078184111	0.991412213740458

We observe that the SVM models overall performes worse than the random forest models in all metrics. The model with radial kernel and c = 10 is the best with respect to these performance metrics, but falls short when compared to the best random forest model. However, looking at the results one can see that the best SVM model is very close to the best random forest models, as far as sensitivity is concerned. As mentioned earlier, the reason why the radial kernel perform worse than the random forest could be because γ is not optimized.

Our modifications of the original data should be taken into account. The overwhelming success of our models might indicate that we took too many liberties in the preproccesing phase. In future work we would experiment with reducing the number of covariates, rather than the number of datapoints. But we thought of this too late. We would then remove `V19` through `V28`, which are weakly correlated with `Class`.

Summary

In this project, we investigated the effectiveness of random forest and SVM methods for predicting credit fraud. Both methods are known for performing well on large datasets with numerous features, which motivated us to explore their potential for fraud detection. Through our analysis, we found that the random forest approach was particularly successful when applied to particular dataset. While the SVM approach with radial kernel performed nearly as well, we were unable to conduct a comprehensive assessment of SVM's parameters due to computational constraints. This may explain why random forest outperformed SVM. Nevertheless, both methods showed reasonable results in terms of sensitivity, which is a crucial metric for fraud detection.