



FRAUD DETECTION IN R

Dealing with imbalanced datasets

Bart Baesens

Professor Data Science at KU Leuven

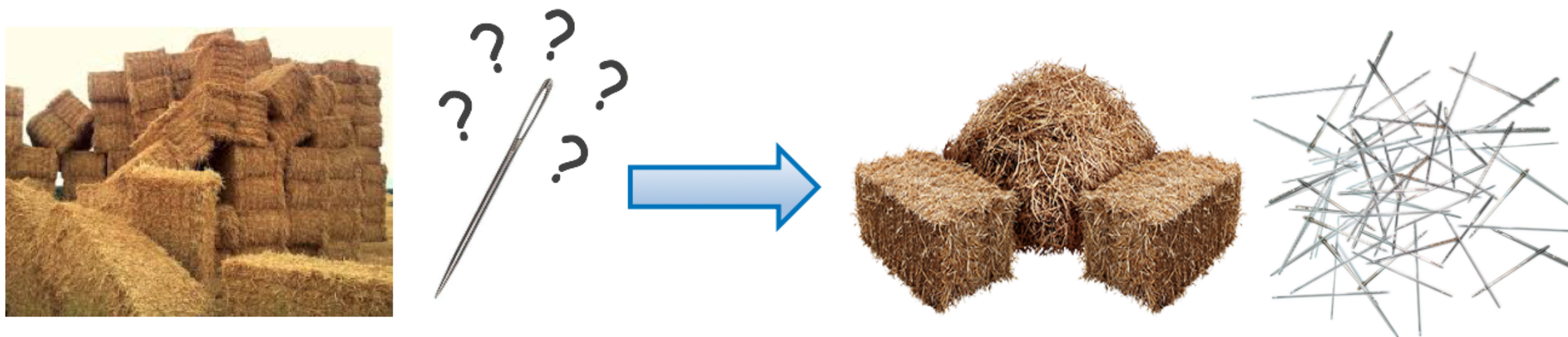
Imbalanced data sets

- **Key challenge:** label events as fraud or not
 - Major challenge for classification methods & anomaly detection techniques
- Classifier tends to favour majority class (= no-fraud)
 - large classification error over the fraud cases
- Classifiers learn better from a *balanced* distribution



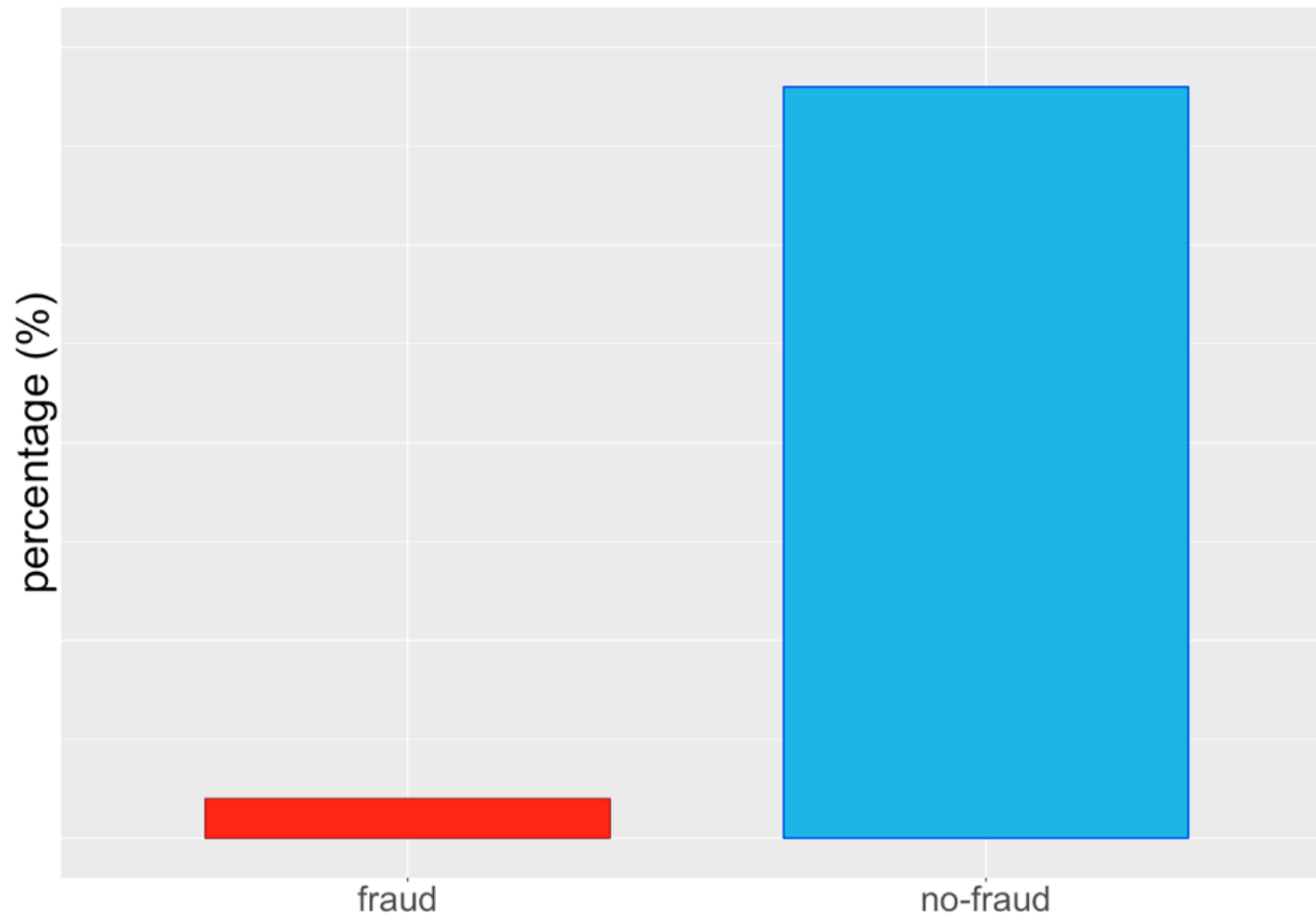
Imbalanced data sets

- **Key challenge** : label events as fraud or not
 - Major challenge for classification methods & anomaly detection techniques
- Classifier tends to favour majority class (= no-fraud)
 - large classification error over the fraud cases
- Classifiers learn better from a *balanced* distribution
- **Possible solution** : *change class distribution* with sampling methods





Original imbalance





Over-sampling minority class...





... or under-sampling majority class ...



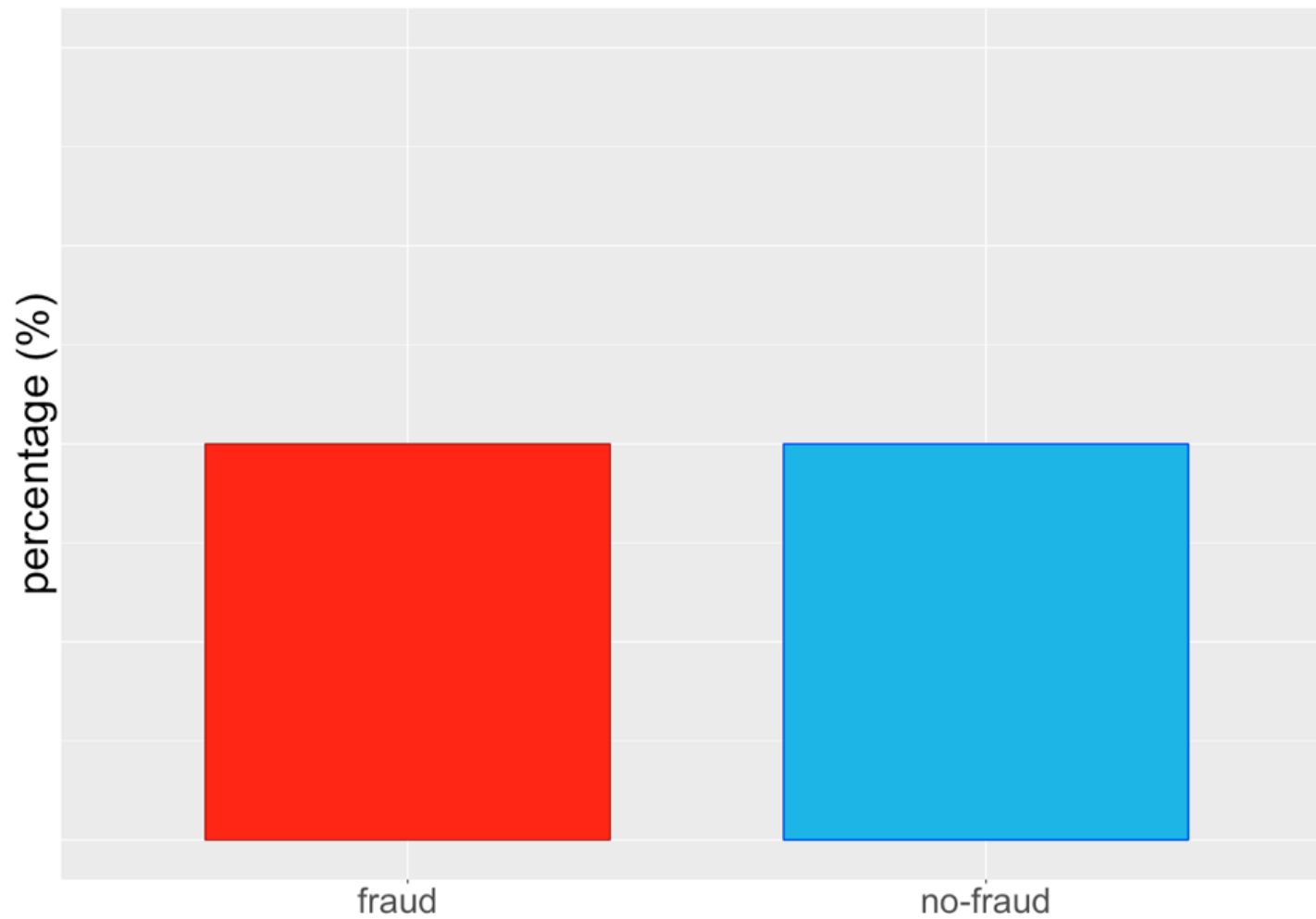


... or both!



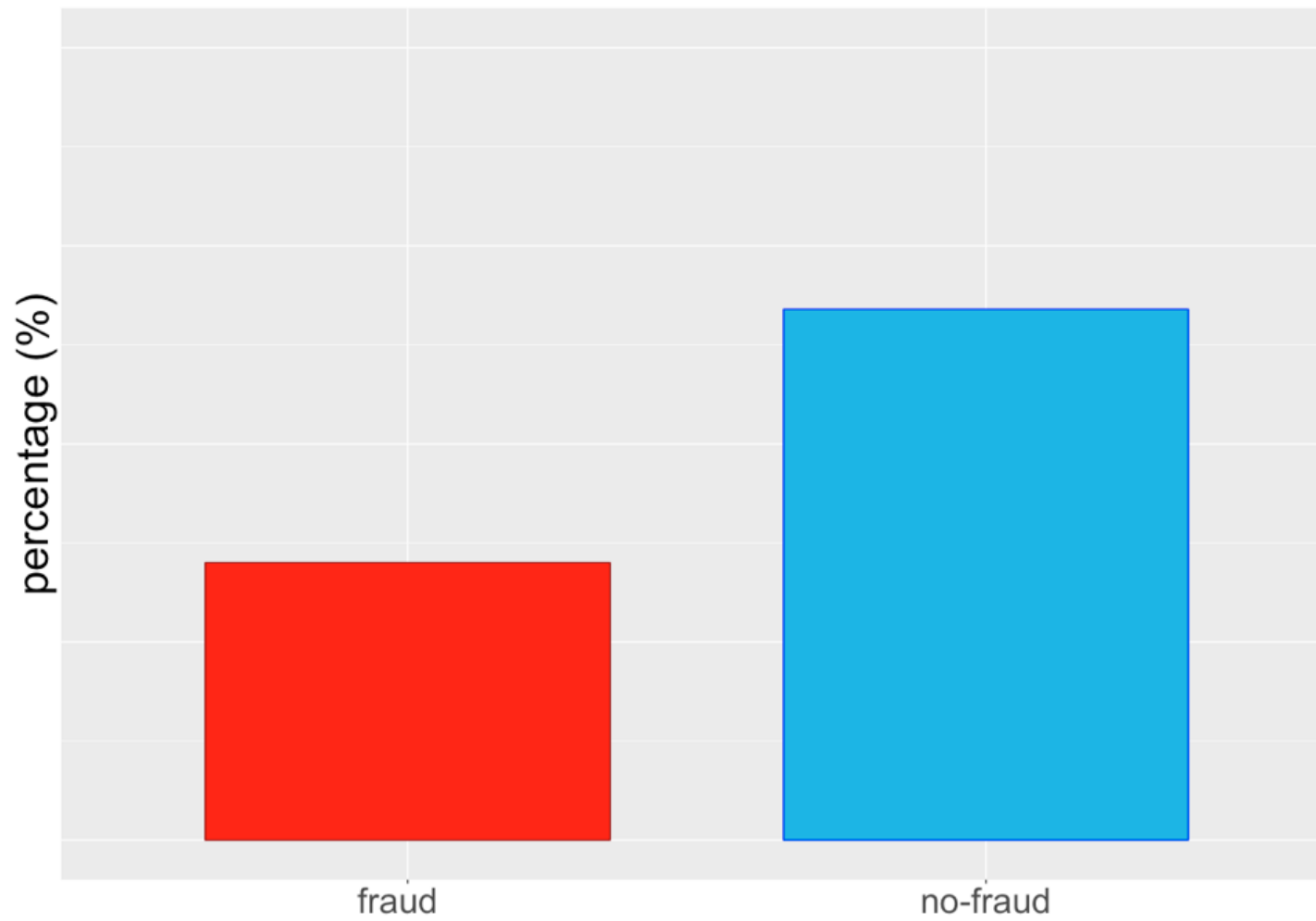


Result after sampling...





... or like this





Random over-sampling (ROS)

Original data

ID	Variables	Class
1	...	Fraud
2	...	No fraud
3	...	No fraud
4	...	Fraud
5	...	No fraud
6	...	No fraud
7	...	No fraud
8	...	No fraud
9	...	Fraud
10	...	No fraud



Random over-sampling (ROS)

Original data

	ID	Variables	Class
Train	1	...	Fraud
	2	...	No fraud
	3	...	No fraud
	4	...	Fraud
	5	...	No fraud
	6	...	No fraud
Test	7	...	No fraud
	8	...	No fraud
	9	...	Fraud
	10	...	No fraud



Random over-sampling (ROS)

Original data

	ID	Variables	Class
Train	1	...	Fraud
	2	...	No fraud
	3	...	No fraud
	4	...	Fraud
	5	...	No fraud
	6	...	No fraud
Test	7	...	No fraud
	8	...	No fraud
	9	...	Fraud
	10	...	No fraud

Random over-sampling (ROS)

Original data

ID	Variables	Class
1	...	Fraud
2	...	No fraud
3	...	No fraud
4	...	Fraud
5	...	No fraud
6	...	No fraud
7	...	No fraud
8	...	No fraud
9	...	Fraud
10	...	No fraud

Train

Test

Over-sampled data

ID	Variables	Class
1	...	Fraud
1	...	Fraud
2	...	No fraud
3	...	No fraud
4	...	Fraud
4	...	Fraud
5	...	No fraud
6	...	No fraud
7	...	No fraud
8	...	No fraud
9	...	Fraud
10	...	No fraud

Train

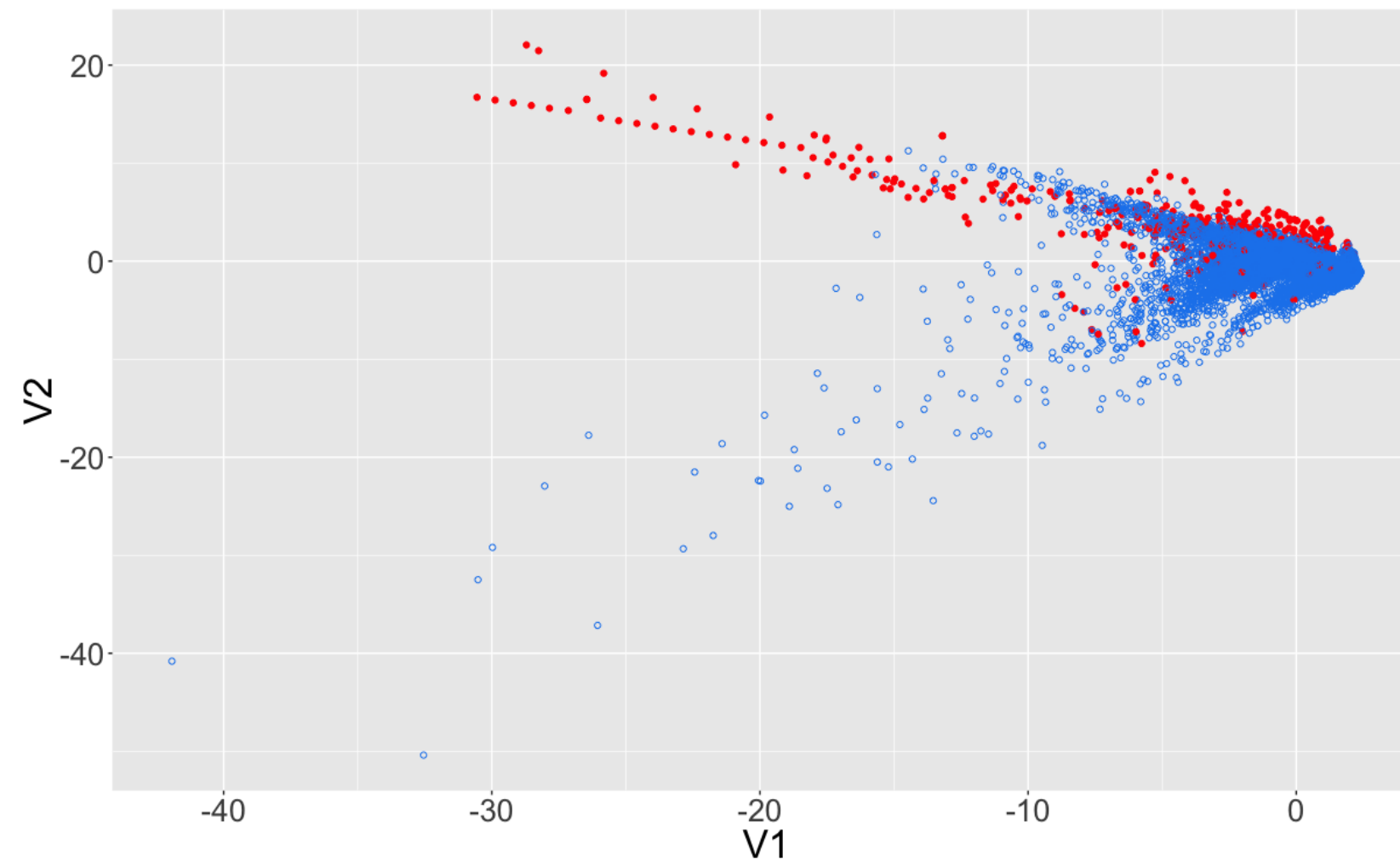
Test



Random over-sampling in practice

- Credit Card Fraud Detection dataset on Kaggle
 - \sim 300K anonymized credit card transfers labeled as fraudulent or genuine
- About the data...
 - Numerical (anonymized) variables: V1, V2, ... , V28
 - Time = seconds elapsed between each transfer and first transfer in dataset
 - Amount = transaction amount
 - Class = response variable: value 1 in case of fraud and 0 otherwise

A look at (a subset of) the dataset



Check the imbalance

```
head(creditcard)
```

	Time	V1	V2	...	V27	V28	Amount	Class
1	0	1.1918571	0.2661507	...	-0.0089830991	0.01472417	2.69	0
2	10	0.3849782	0.6161095	...	0.0424724419	-0.05433739	9.99	0
3	12	-0.7524170	0.3454854	...	-0.1809975001	0.12939406	15.99	0
4	17	0.9624961	0.3284610	...	0.0163706433	-0.01460533	34.09	0
5	34	0.2016859	0.4974832	...	0.1427572469	0.21923761	9.99	0
6	35	1.3863970	-0.7942095	...	0.0005313319	0.01991062	30.90	0

```
table(creditcard$Class)
```

0	1
24108	492

```
prop.table(table(creditcard$Class))
```

0	1
0.98	0.02

ovun.sample from ROSE package

- ROSE package: Random Over-Sampling Examples
- `ovun.sample()` for random over-sampling, under-sampling or combination!

```
n_legit <- 24108
new_frac_legit <- 0.50
new_n_total <- n_legit/new_frac_legit # = 24108/0.50 = 48216

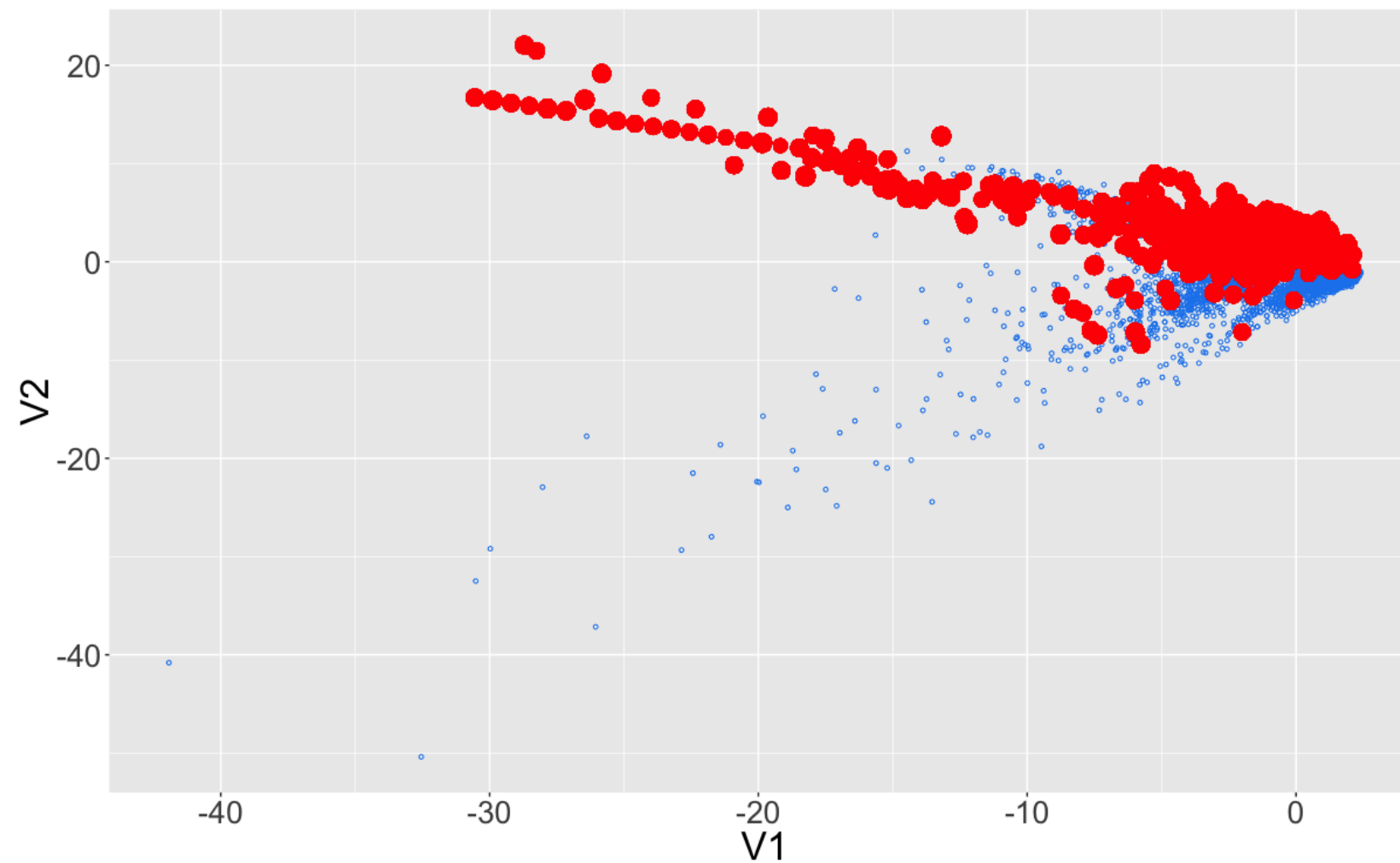
library(ROSE)
oversampling_result <- ovun.sample(Class ~ .,
                                   data = creditcard,
                                   method = "over",
                                   N = new_n_total,
                                   seed = 2018)

oversampled_credit <- oversampling_result$data

table(oversampled_credit$Class)

      0      1
24108 24108
```

A look at the over-sampled dataset





FRAUD DETECTION IN R

Let's practice!



FRAUD DETECTION IN R

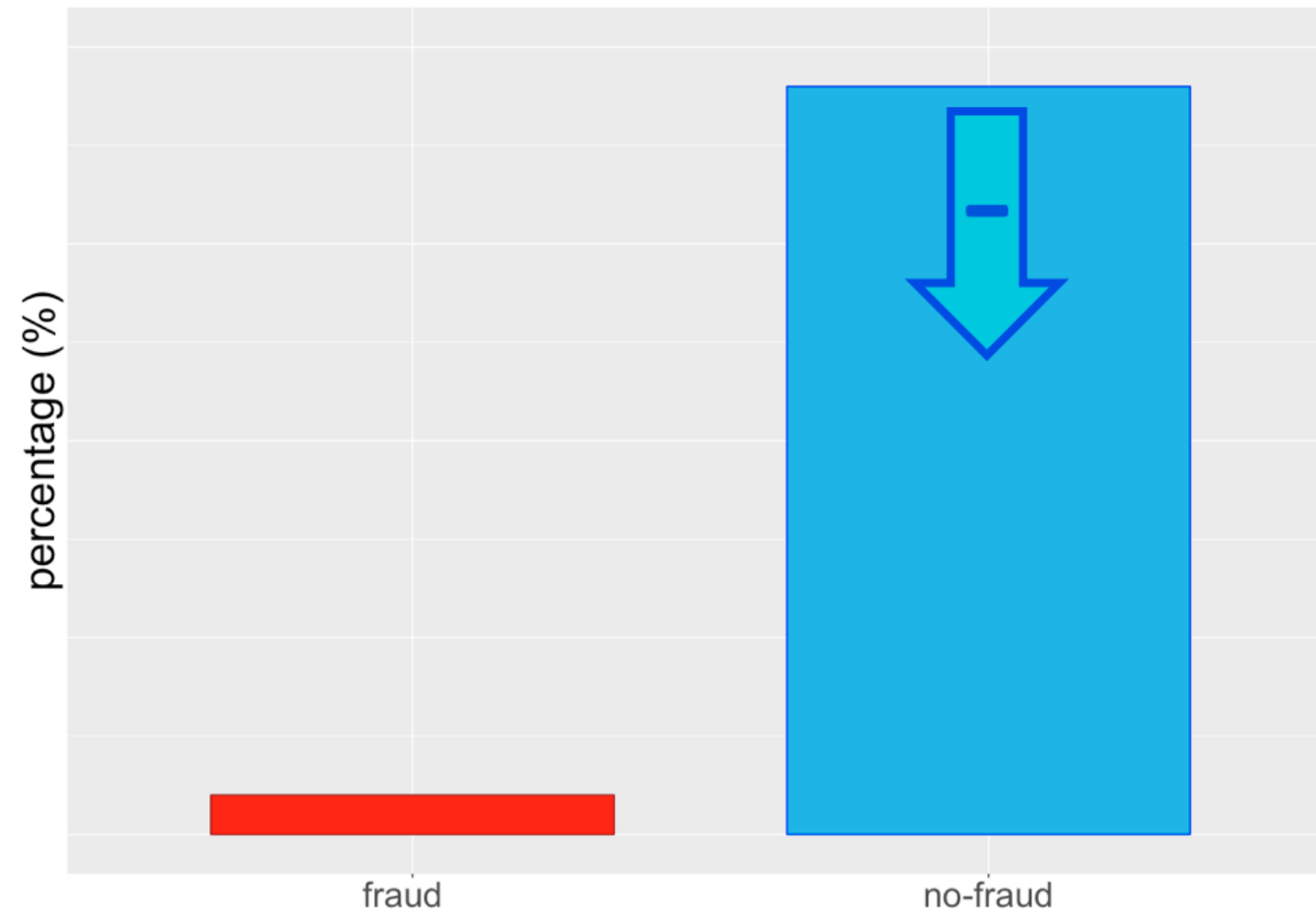
Random under-sampling

Bart Baesens

Professor Data Science at KU Leuven



Random under-sampling (RUS)



Random under-sampling (RUS)

Original data

	ID	Variables	Class
Train	1	...	Fraud
	2	...	No fraud
	3	...	No fraud
	4	...	Fraud
	5	...	No fraud
	6	...	No fraud
Test	7	...	No fraud
	8	...	No fraud
	9	...	Fraud
	10	...	No fraud



Random under-sampling (RUS)

Original data

	ID	Variables	Class
Train	1	...	Fraud
	2	...	No-Fraud
	3	...	No fraud
	4	...	Fraud
	5	...	No-Fraud
	6	...	No fraud
Test	7	...	No fraud
	8	...	No fraud
	9	...	Fraud
	10	...	No fraud

Random under-sampling (RUS)

Original data

ID	Variables	Class
1	...	Fraud
2	...	No Fraud
3	...	No fraud
4	...	Fraud
5	...	No Fraud
6	...	No fraud
7	...	No fraud
8	...	No fraud
9	...	Fraud
10	...	No fraud

Train

Test

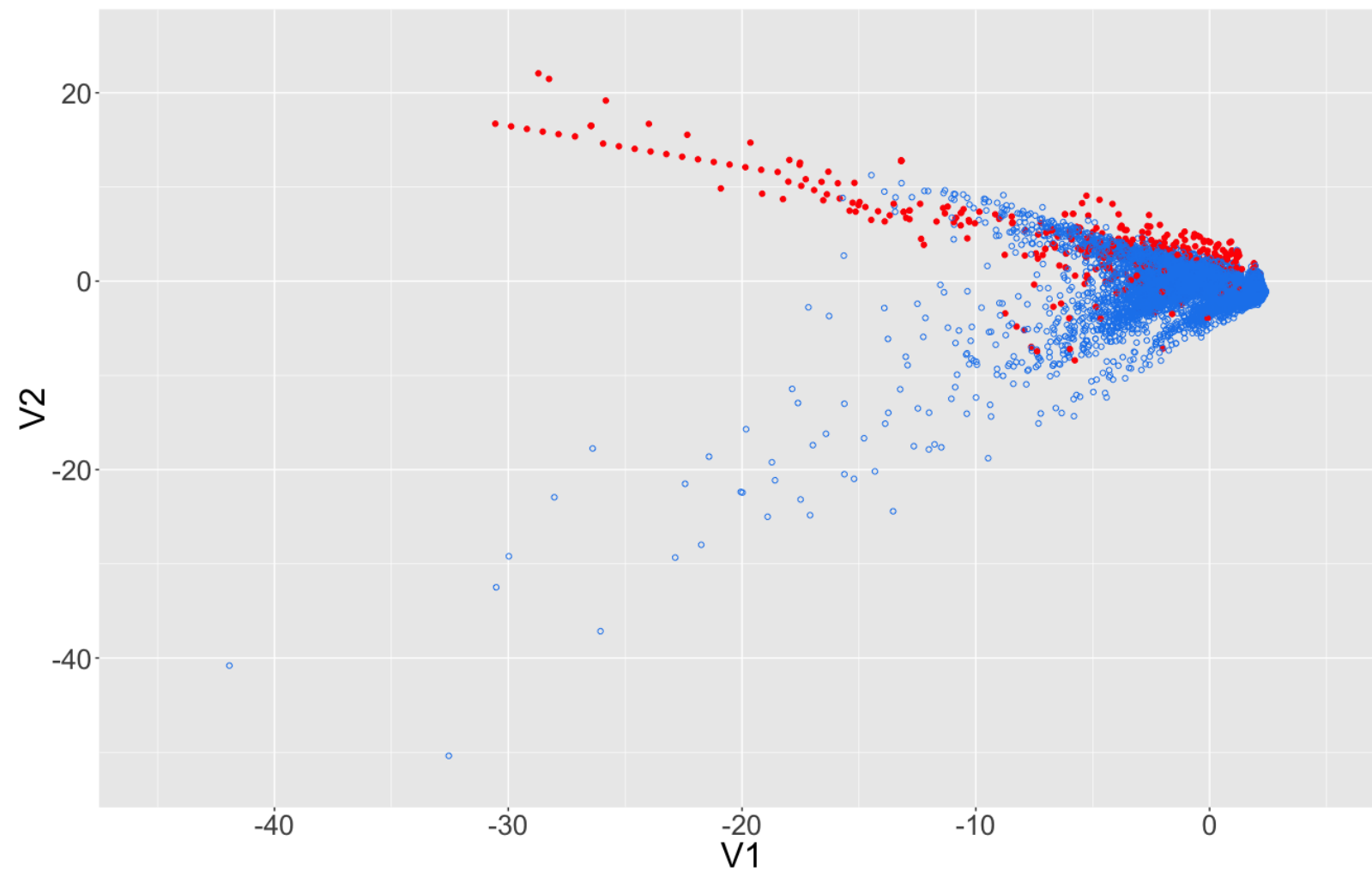
Under-sampled data

ID	Variables	Class
1	...	Fraud
3	...	No fraud
4	...	Fraud
6	...	No fraud
7	...	No fraud
8	...	No fraud
9	...	Fraud
10	...	No fraud

Train

Test

A look at the imbalanced dataset



Again ovun.sample

- `ovun.sample()` from ROSE package also for random under-sampling!

```
table(creditcard$Class)

  0    1
24108 492

n_fraud <- 492
new_frac_fraud <- 0.50
new_n_total <- n_fraud/new_frac_fraud # = 492/0.50 = 984

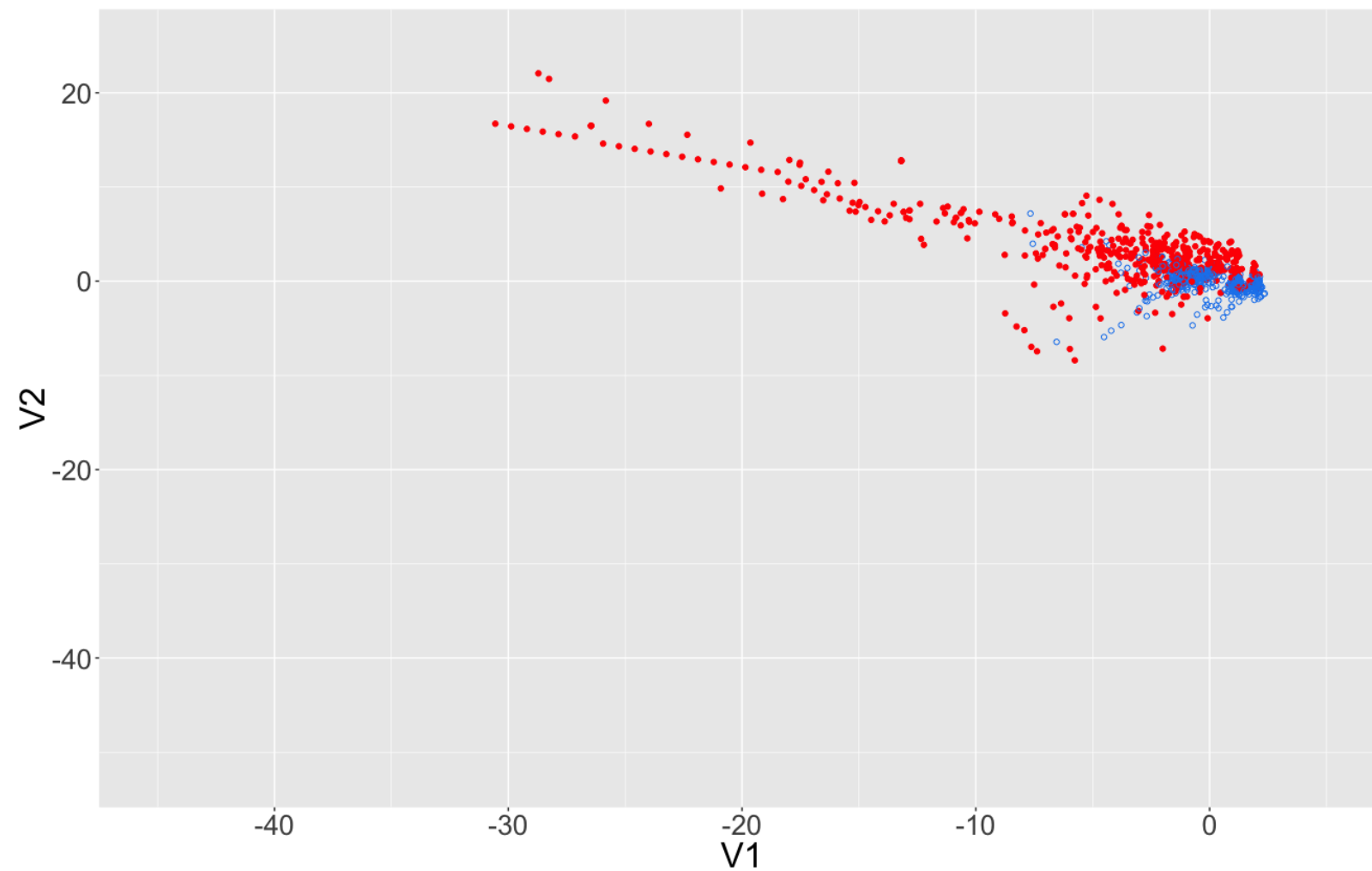
library(ROSE)
undersampling_result <- ovun.sample(Class ~ .,
                                   data = creditcard,
                                   method = "under",
                                   N = new_n_total,
                                   seed = 2018)

undersampled_credit <- undersampling_result$data

table(undersampled_credit$Class)

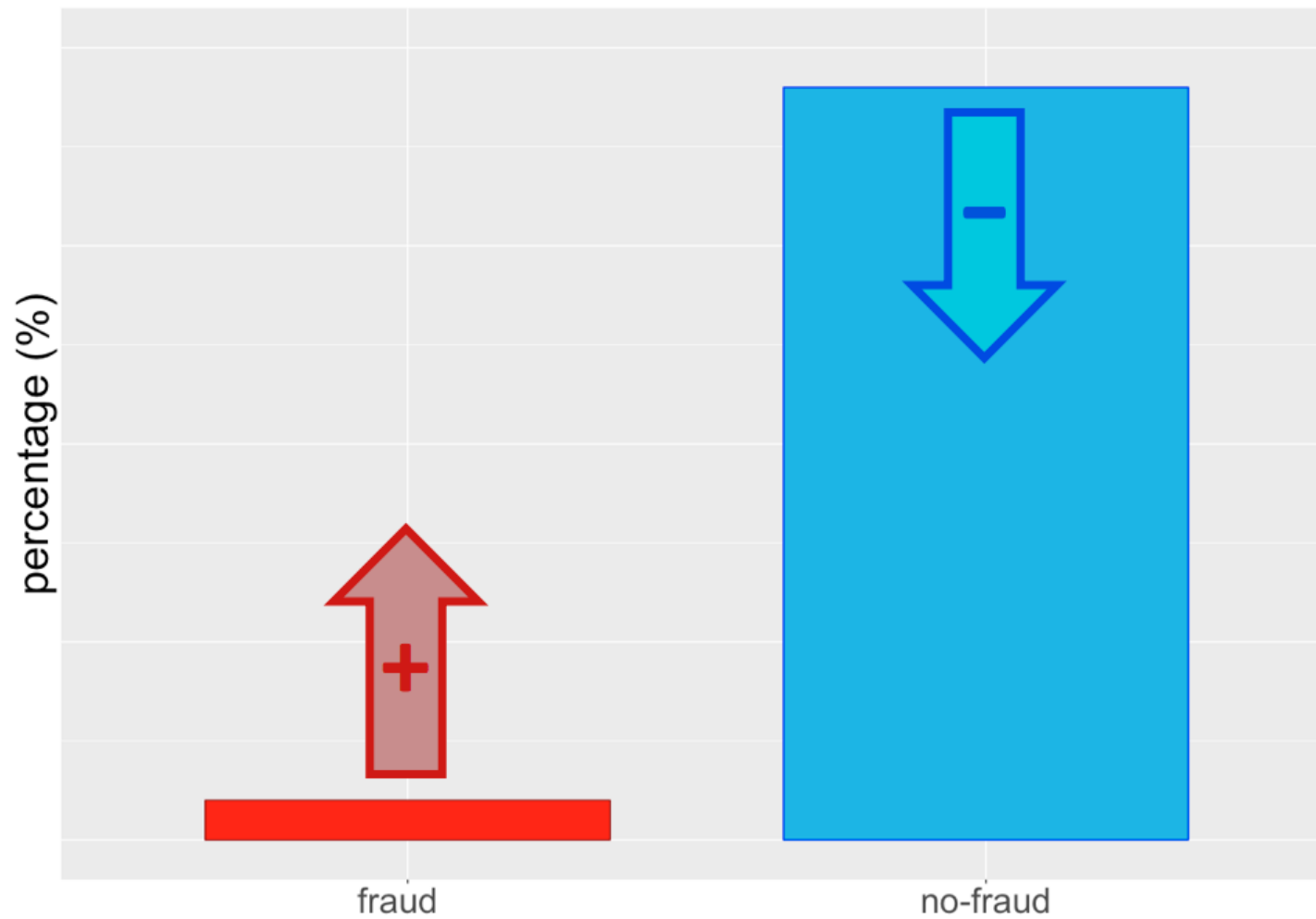
  0    1
492 492
```

A look at the under-sampled dataset





Let's do both!





Combination of over- & under-sampling

```
n_new <- nrow(creditcard) # = 24600
fraction_fraud_new <- 0.50

sampling_result <- ovun.sample(Class ~ .,
                                data = creditcard,
                                method = "both",
                                N = n_new,
                                p = fraction_fraud_new,
                                seed = 2018)

sampled_credit <- sampling_result$data

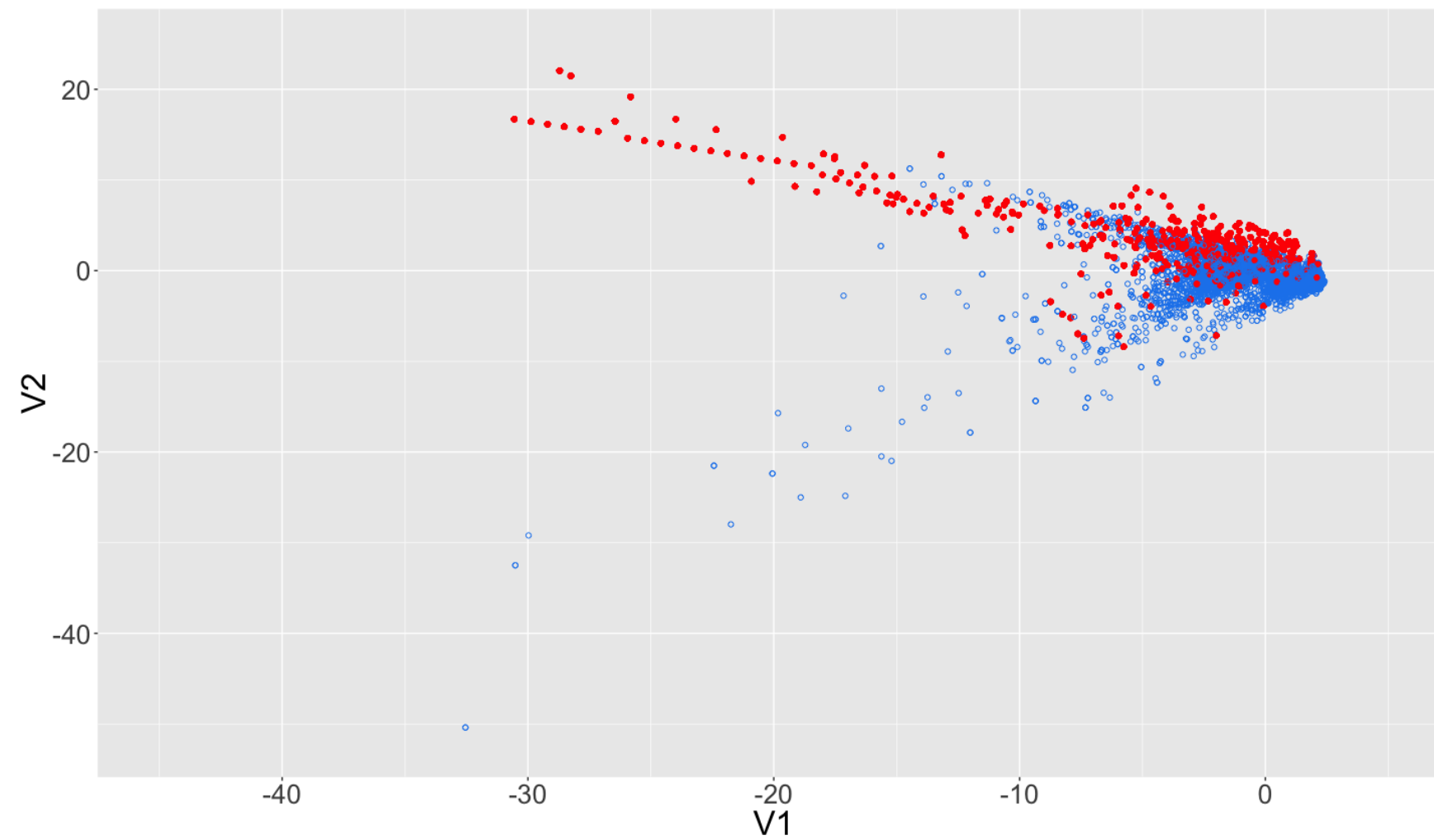
table(sampled_credit$Class)

      0      1
12398 12202

prop.table(table(sampled_credit$Class))

      0      1
0.5039837 0.4960163
```

Result!





FRAUD DETECTION IN R

Let's practice!



FRAUD DETECTION IN R

Synthetic Minority Over-sampling

Sebastiaan Höppner

PhD researcher in Data Science at KU Leuven



Over-sampling with 'SMOTE'

- **SMOTE** : **S**ynthetic **M**inority **O**versampling **T**Echnique (Chawla et al., 2002)
- Over-sample minority class (i.e. fraud) by creating synthetic minority cases



Example: credit transfer data

```
dim(transfer_data)
```

```
[1] 1000    4
```

```
head(transfer_data)
```

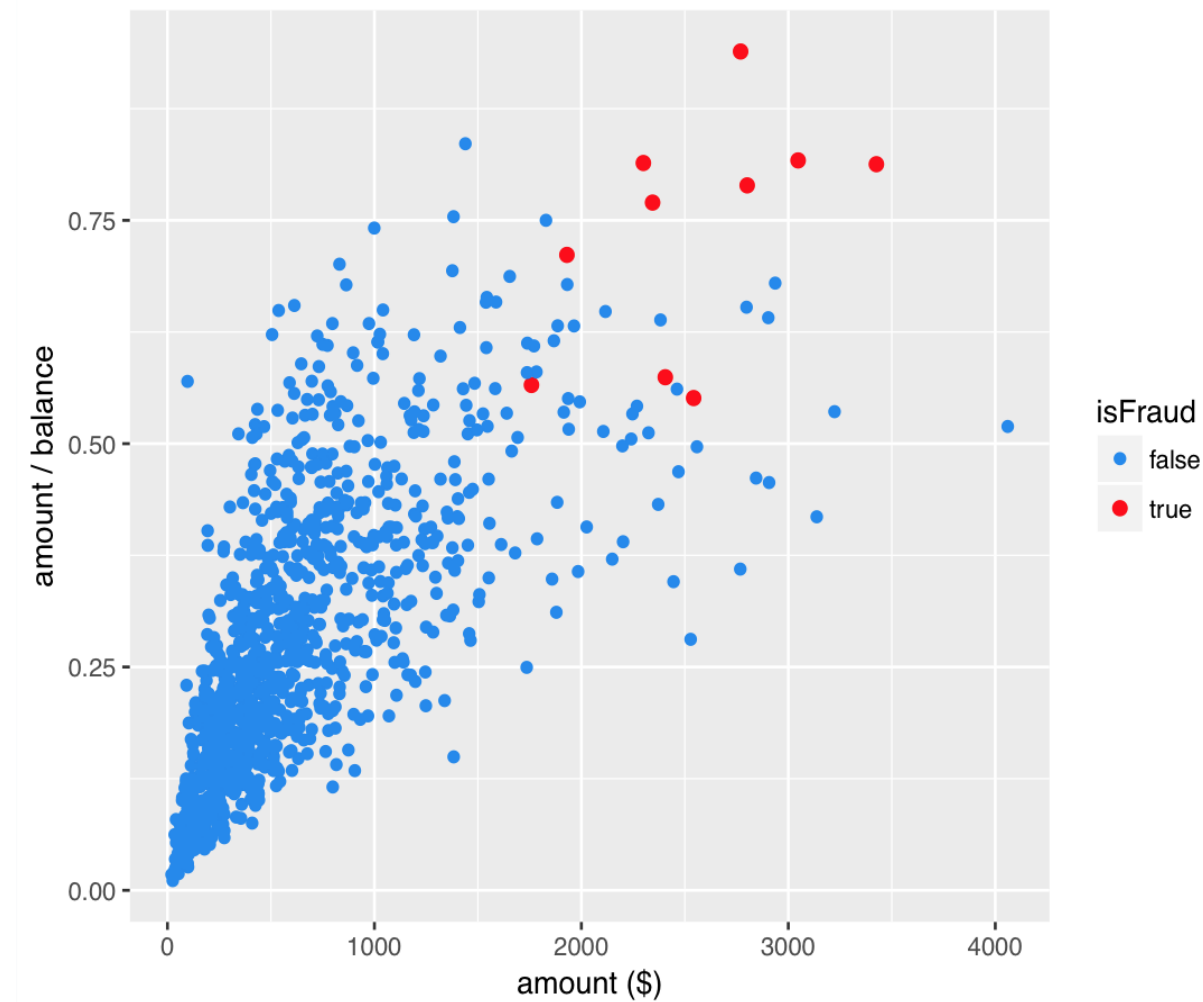
	isFraud	amount	balance	ratio
1	false	528.6840	1529.4732	0.3456641
2	false	184.0193	836.3509	0.2200265
3	false	1885.8024	2984.0684	0.6319568
4	false	732.0286	1248.7217	0.5862224
5	false	694.0790	1464.3630	0.4739801
6	false	2461.9941	4387.8114	0.5610984

```
prop.table(table(transfer_data$isFraud))
```

false	true
0.99	0.01

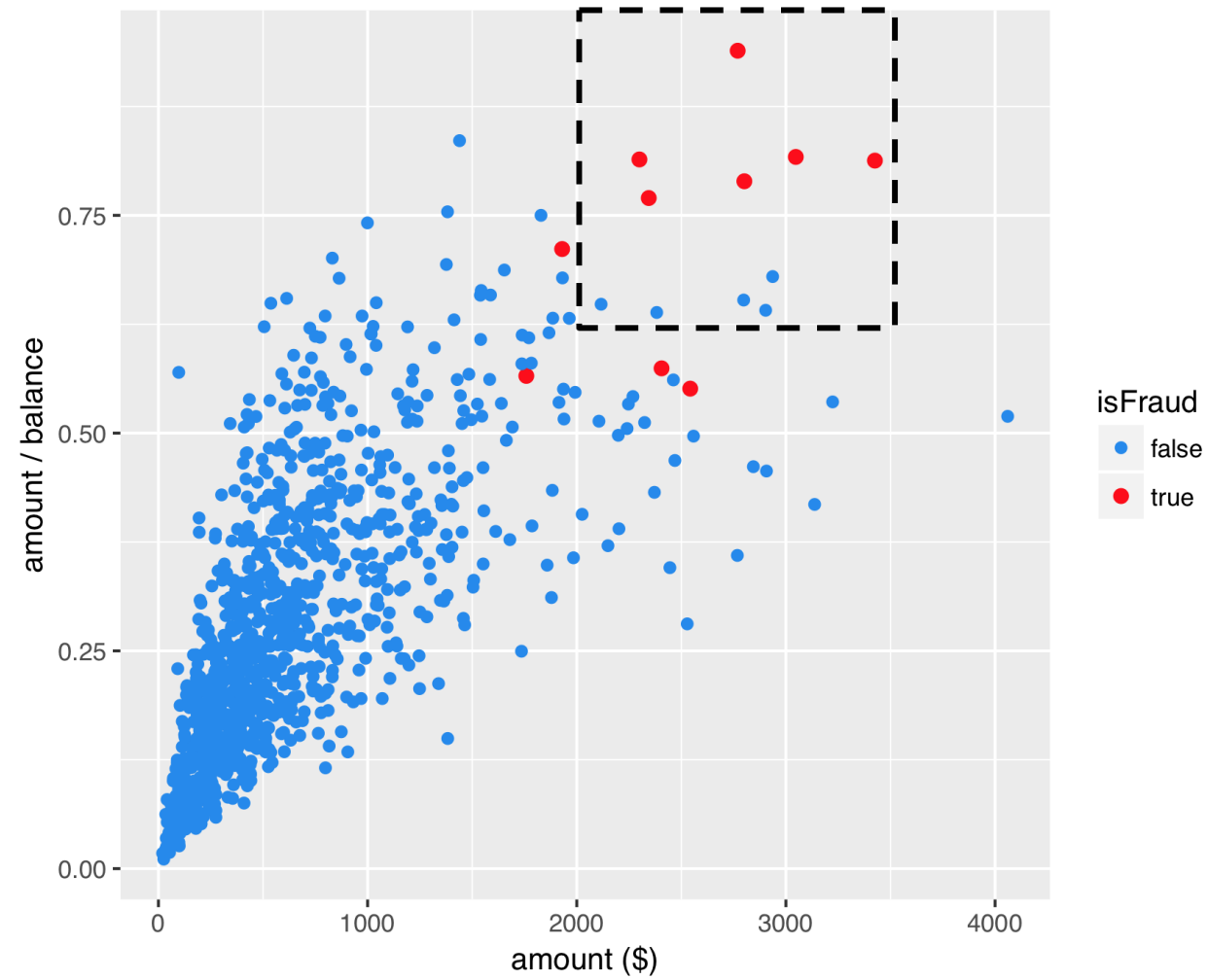


Look at the data (ratio vs amount)





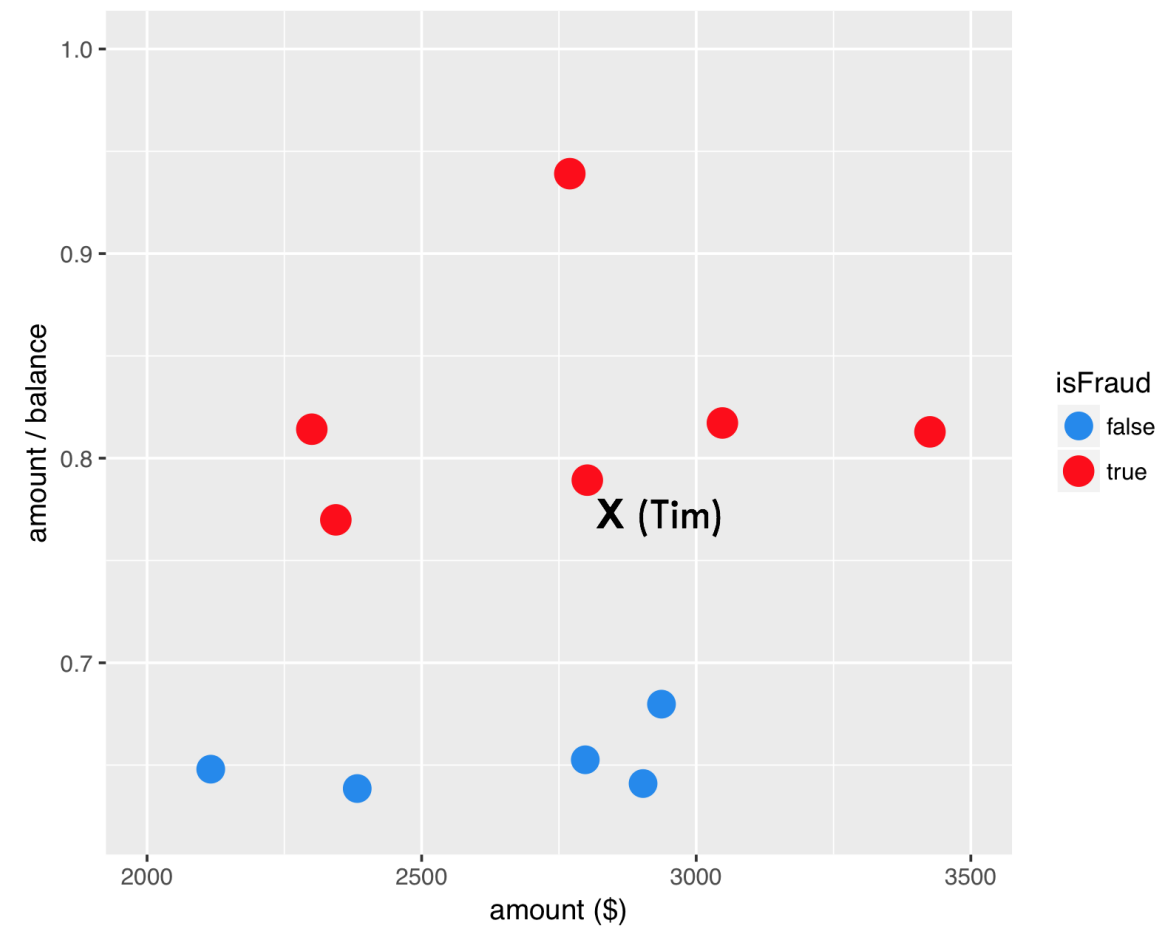
Focus on fraud cases





SMOTE

Let's select a fraud case **X** (Tim)



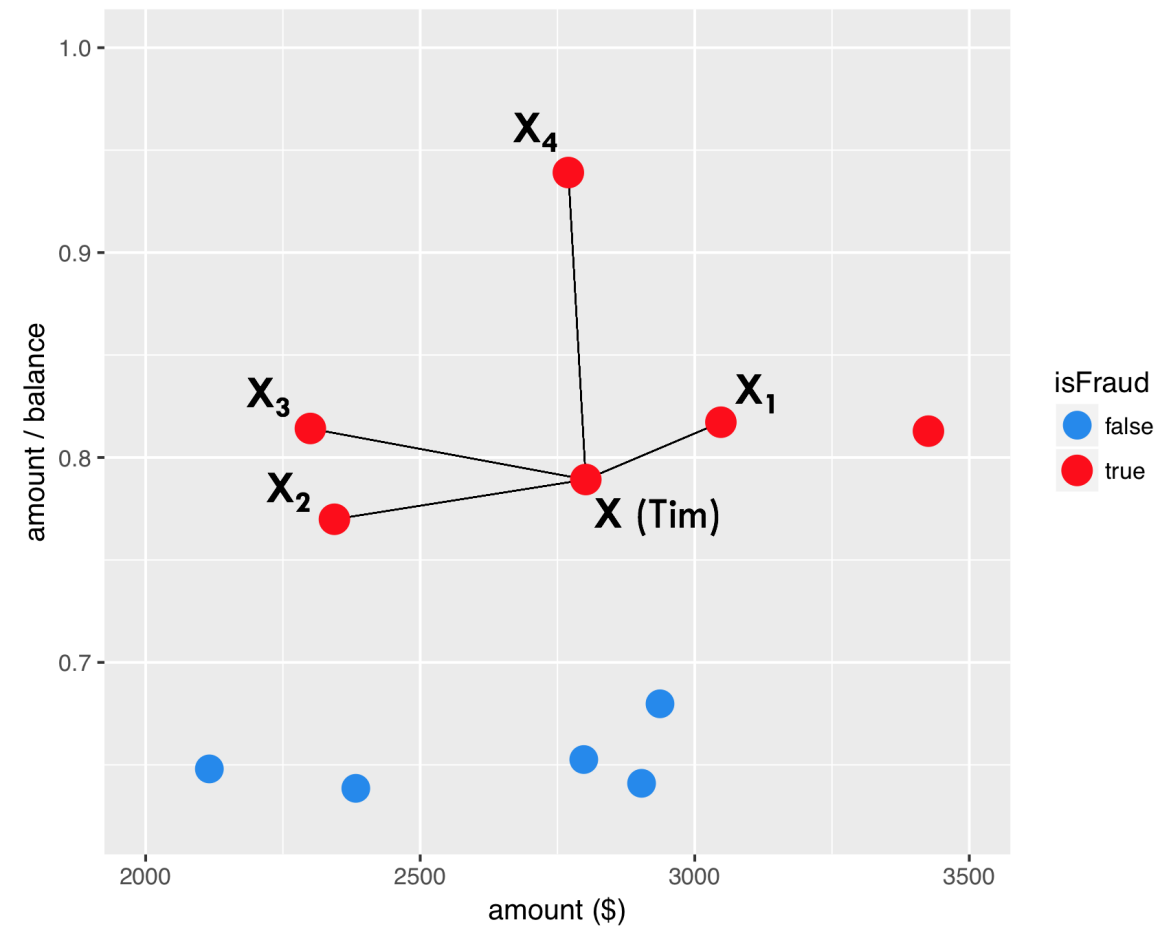


SMOTE - step 1

Step 1

Find K nearest fraudulent neighbors of **X** (Tim)

e.g. $K = 4$

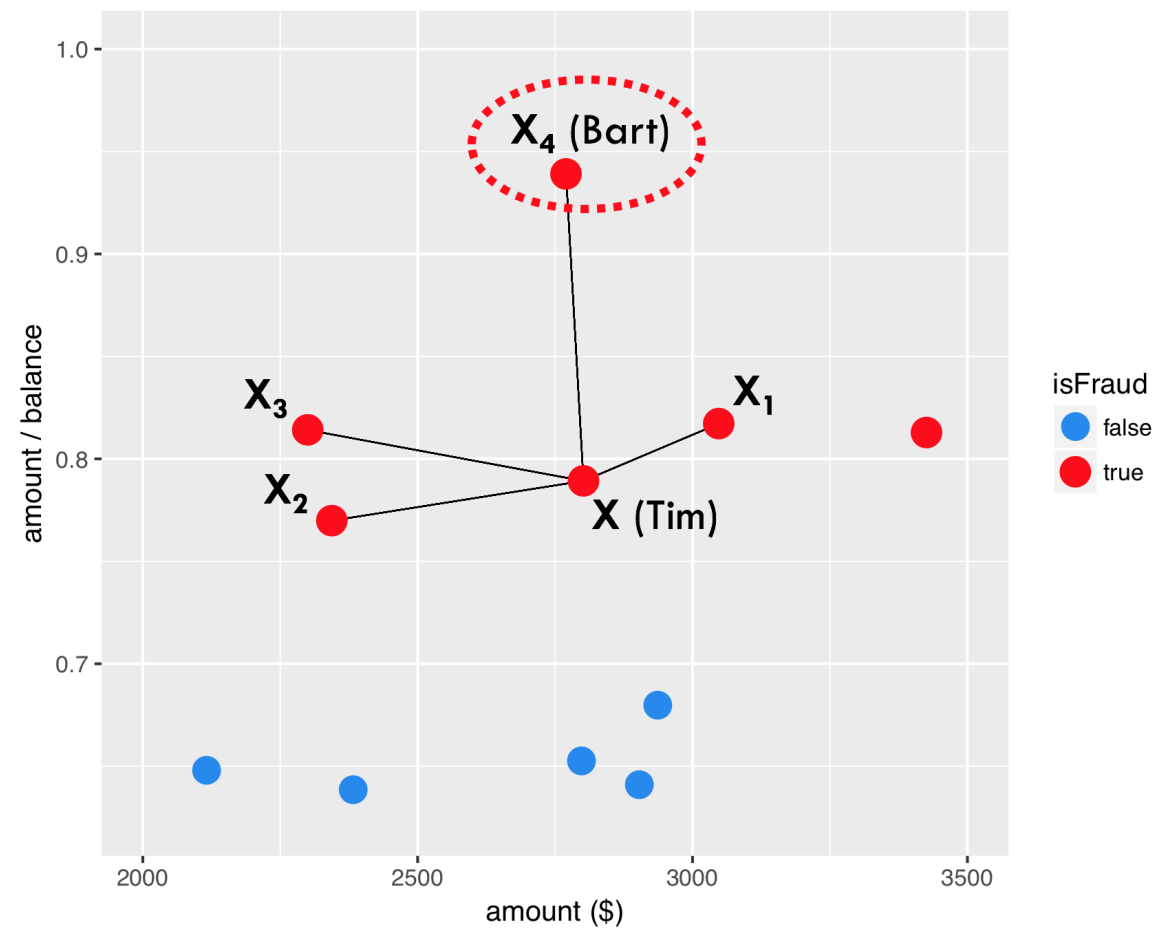


SMOTE - step 2

Step 2

Randomly choose one of Tim's nearest neighbors

e.g. **X₄** (Bart)





SMOTE - step 3

Step 3 : create synthetic sample

X (Tim)

Amount	Ratio
2800	0.79

X₄ (Bart)

Amount	Ratio
2770	0.94



SMOTE - step 3

Step 3 : create synthetic sample

X (Tim)

Amount	Ratio
2800	0.79

X₄ (Bart)

Amount	Ratio
2770	0.94

Choose random number
between 0 and 1, e.g. **0.6**

SMOTE - step 3

Step 3 : create synthetic sample

X (Tim)

Amount	Ratio
2800	0.79

X₄ (Bart)

Amount	Ratio
2770	0.94

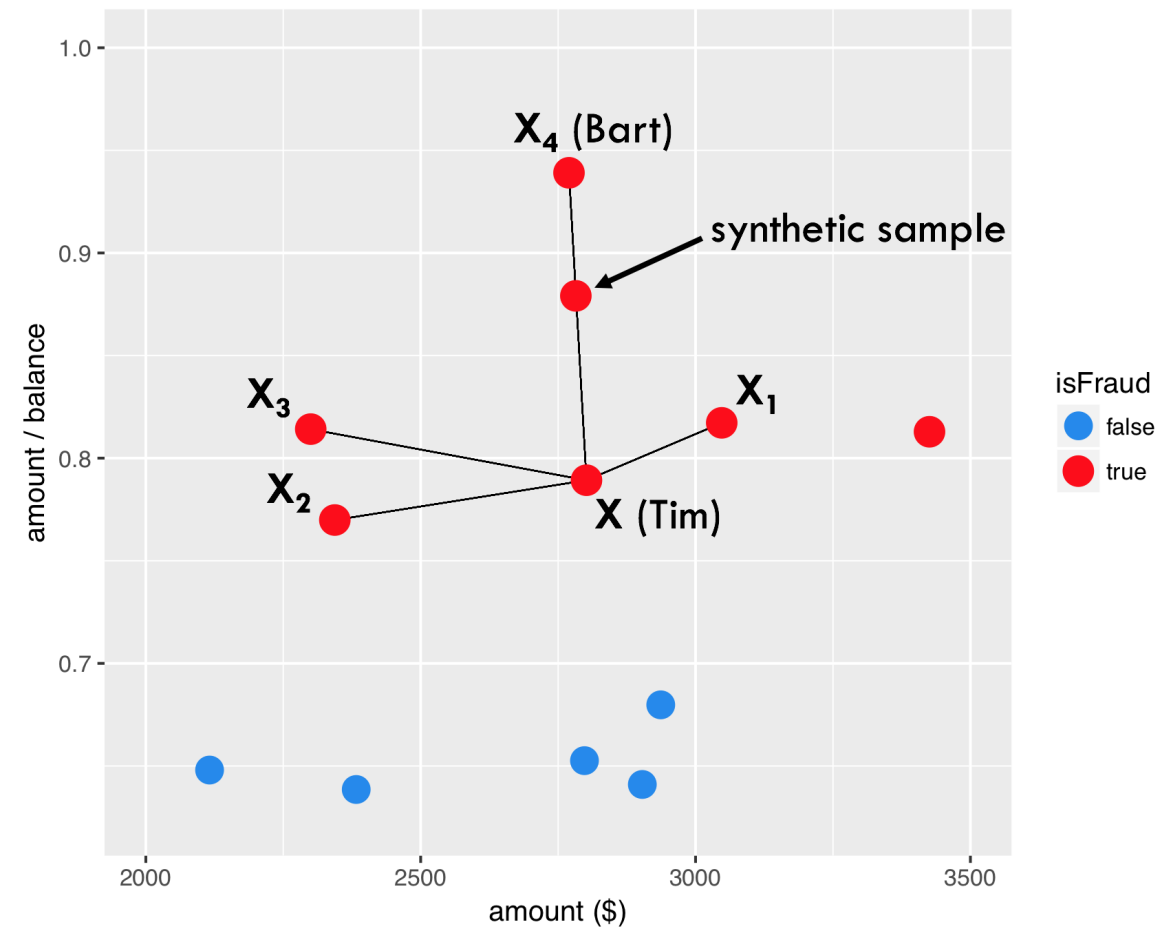
Choose random number
between 0 and 1, e.g. **0.6**

Synthetic sample

Amount	Ratio
$2800 + 0.6 * (2770 - 2800)$ = 2782	$0.79 + 0.6 * (0.94 - 0.79)$ = 0.88



SMOTE - step 3



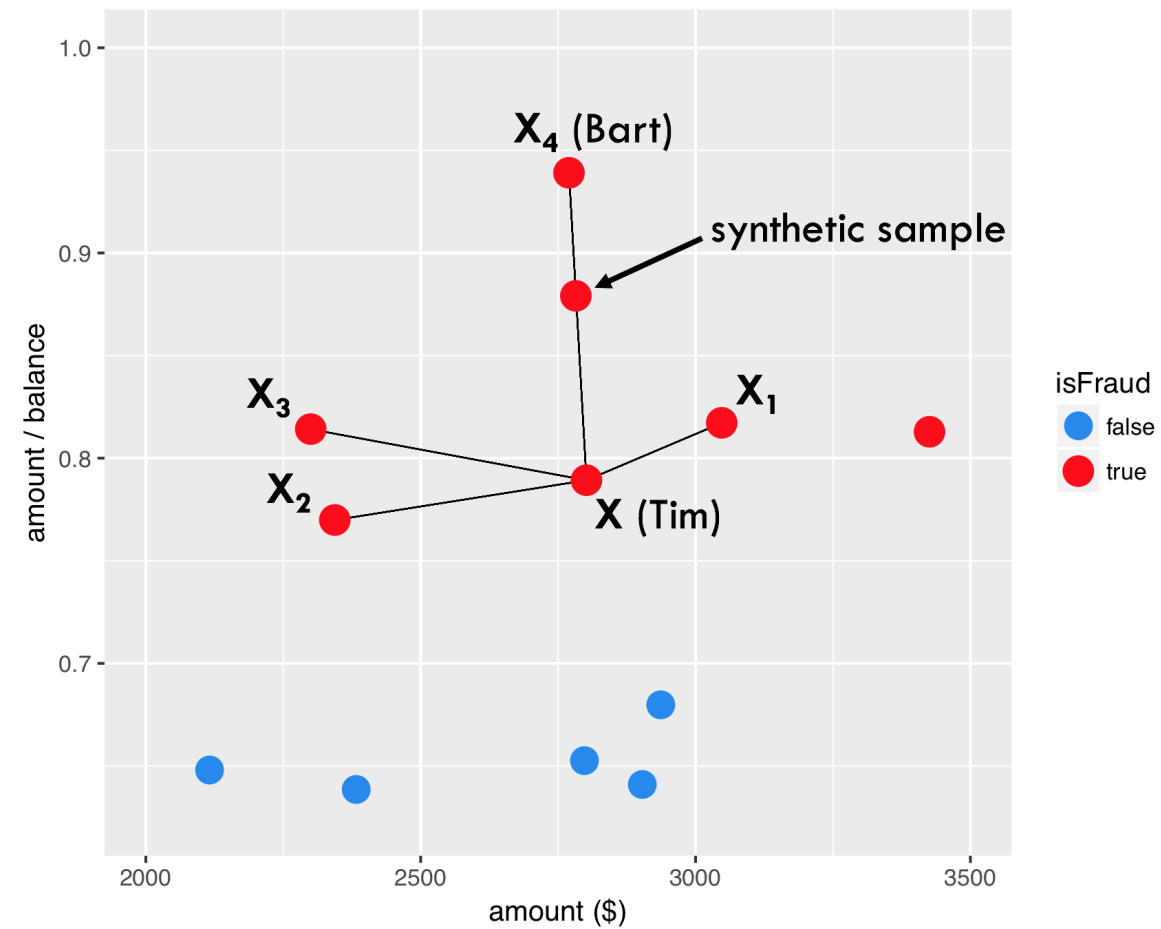


SMOTE - step 4

Step 4

Repeat steps 1-3 for each fraud case
`dup_size` times

e.g. `dup_size = 10`



SMOTE on transfer_data

```
> library(smotefamily)

> smote_output = SMOTE(X = transfer_data[, -1],
                       target = transfer_data$isFraud,
                       K = 4,
                       dup_size = 10)

> oversampled_data = smote_output$data

> table(oversampled_data$isFraud)

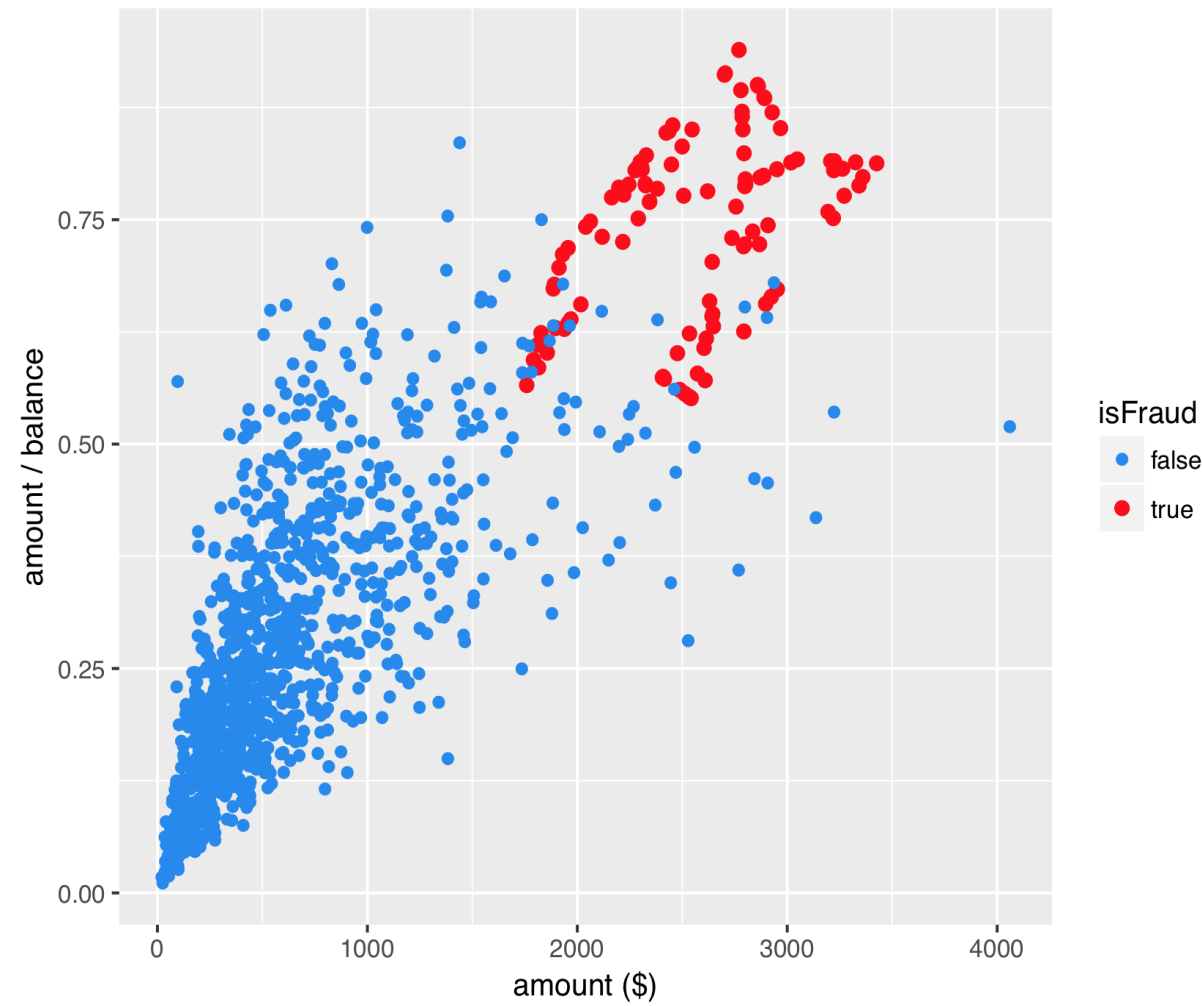
false  true
  990   110

> prop.table(table(oversampled_data$isFraud))

false  true
  0.9   0.1
```



Synthetic fraud cases





FRAUD DETECTION IN R

Let's practice!



FRAUD DETECTION IN R

From dataset to detection model

Sebastiaan Höppner

PhD researcher in Data Science at KU Leuven



Roadmap

- (1) Divide dataset in **training set** and **test set**
- (2) **Choose** a machine learning **model**
- (3) **Apply SMOTE** on training set to balance the class distribution
- (4) **Train model** on re-balanced training set
- (5) **Test performance** on (original) test set



Divide dataset in training & set

- Split the dataset into a **training set** and a **test set** (e.g. 50/50, 75/25, ...)
- Make sure that both sets have identical class distribution (at first)
- Example: 50% training set and 50% test set

```
prop.table(table(train$Class))
```

```
  0    1  
0.98 0.02
```

```
prop.table(table(test$Class))
```

```
  0    1  
0.98 0.02
```



Choose & train machine learning model

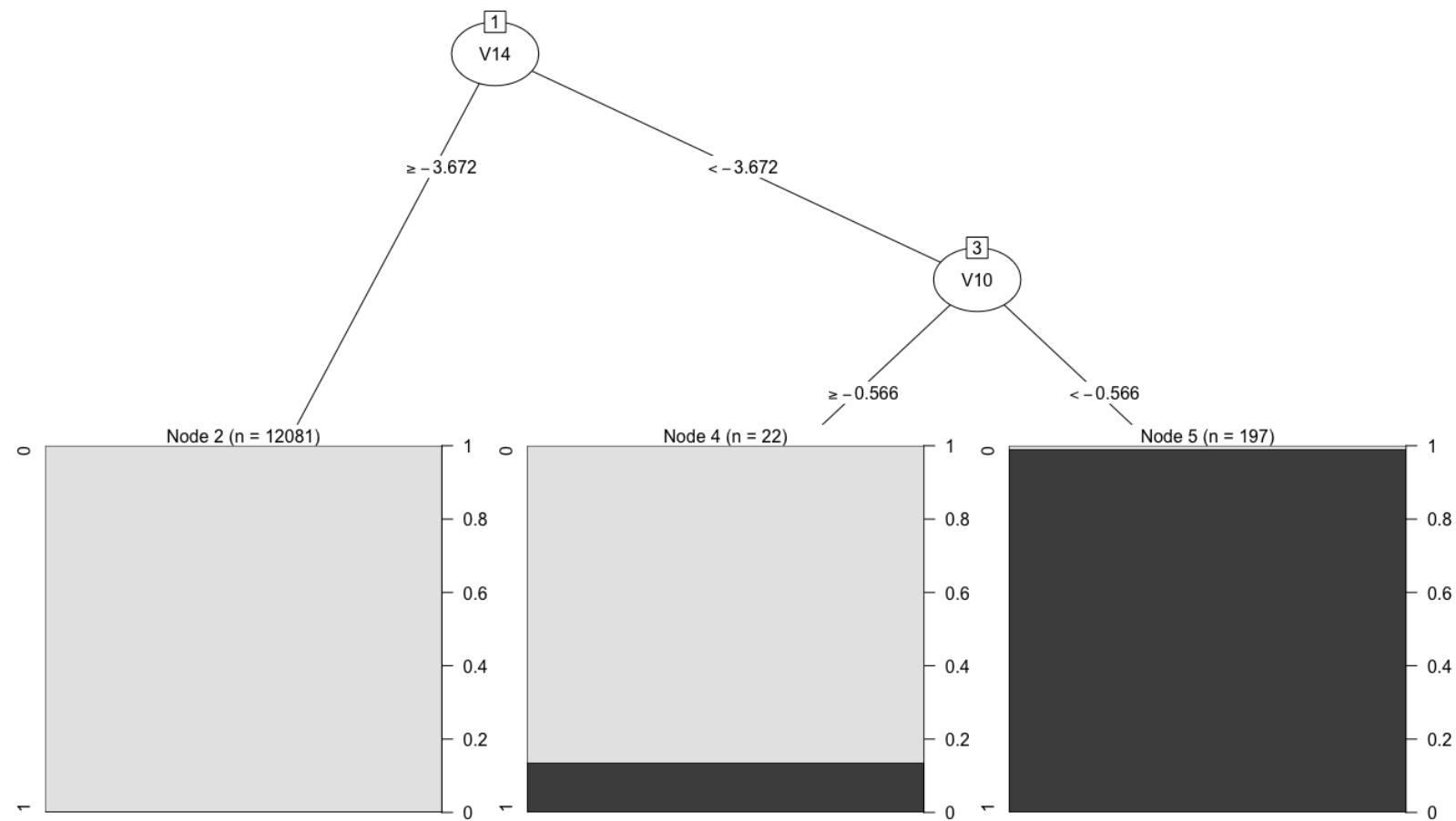
- Decision tree, artificial neural network, support vector machines, logistic regression, random forest, Naive Bayes, k-Nearest Neighbors, ...
- **Example:** Classification And Regression Tree (**CART**) algorithm
- Function `rpart` in `rpart` package

```
library(rpart)

model1 = rpart(Class ~ ., data = train)
```

A simple classification tree model

```
library(partykit)
plot(as.party(model1))
```





Test performance on test set

```
# Predict fraud probability
scores1 = predict(model1, newdata = test, type = "prob")[, 2]

# Predict class (fraud or not)
predicted_class1 = factor(ifelse(scores1 > 0.5, 1, 0))

# Confusion matrix & accuracy
library(caret)
CM1 = confusionMatrix(data = predicted_class1, reference = test$Class)

CM1
```

	Reference	
Prediction	0	1
0	12046	55
1	8	191

```
Accuracy : 0.994878

# Area Under ROC Curve (AUC)
library(pROC)
auc(roc(response = test$Class, predictor = scores1))

Area under the curve: 0.8938
```



Apply SMOTE on training set

```
library(smotefamily)
set.seed(123)

smote_result = SMOTE(X = train[, -17],
                     target = train$Class,
                     K = 5,
                     dup_size = 10)

train_oversampled = smote_result$data
colnames(train_oversampled)[17] = "Class"

table(train_oversampled$Class)

      0      1
12054  2706

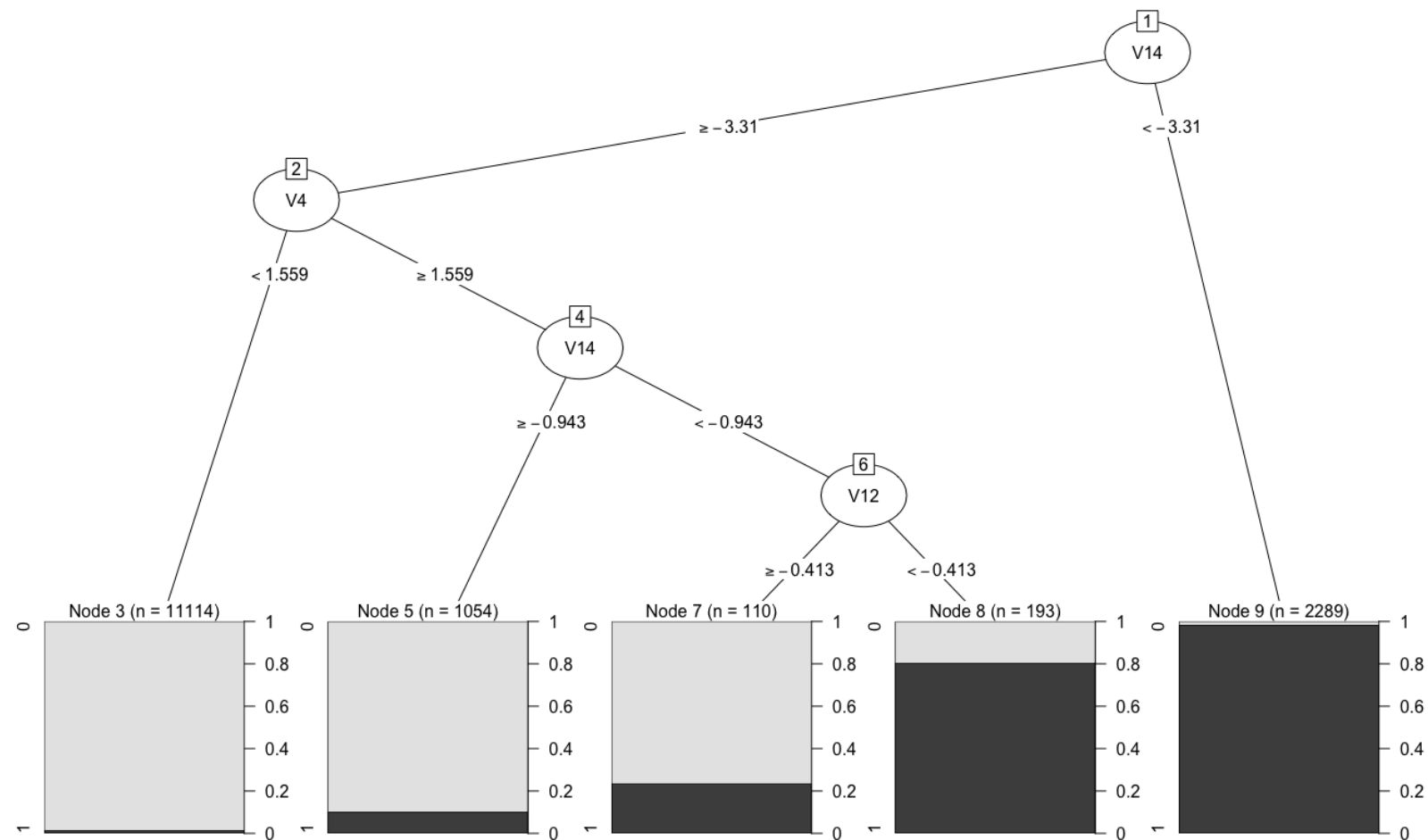
prop.table(table(train_oversampled$Class))

      0      1
0.8166667 0.1833333
```

Train model on re-balanced training set

```
library(rpart)

model2 = rpart(Class ~ ., data = train_oversampled)
```



Test performance of new model on test set

```
# Predict fraud probability
scores2 = predict(model2, newdata = test, type = "prob")[, 2]

# Predict class (fraud or not)
predicted_class2 = factor(ifelse(scores2 > 0.5, 1, 0))

# Confusion matrix & accuracy
library(caret)
CM2 = confusionMatrix(data = predicted_class2, reference = test$Class)

CM2
```

	Reference	
Prediction	0	1
0	11967	34
1	87	212

```
Accuracy : 0.9901626

# Area Under ROC Curve (AUC)
library(pROC)
auc(roc(response = test$Class, predictor = scores2))

Area under the curve: 0.9538
```

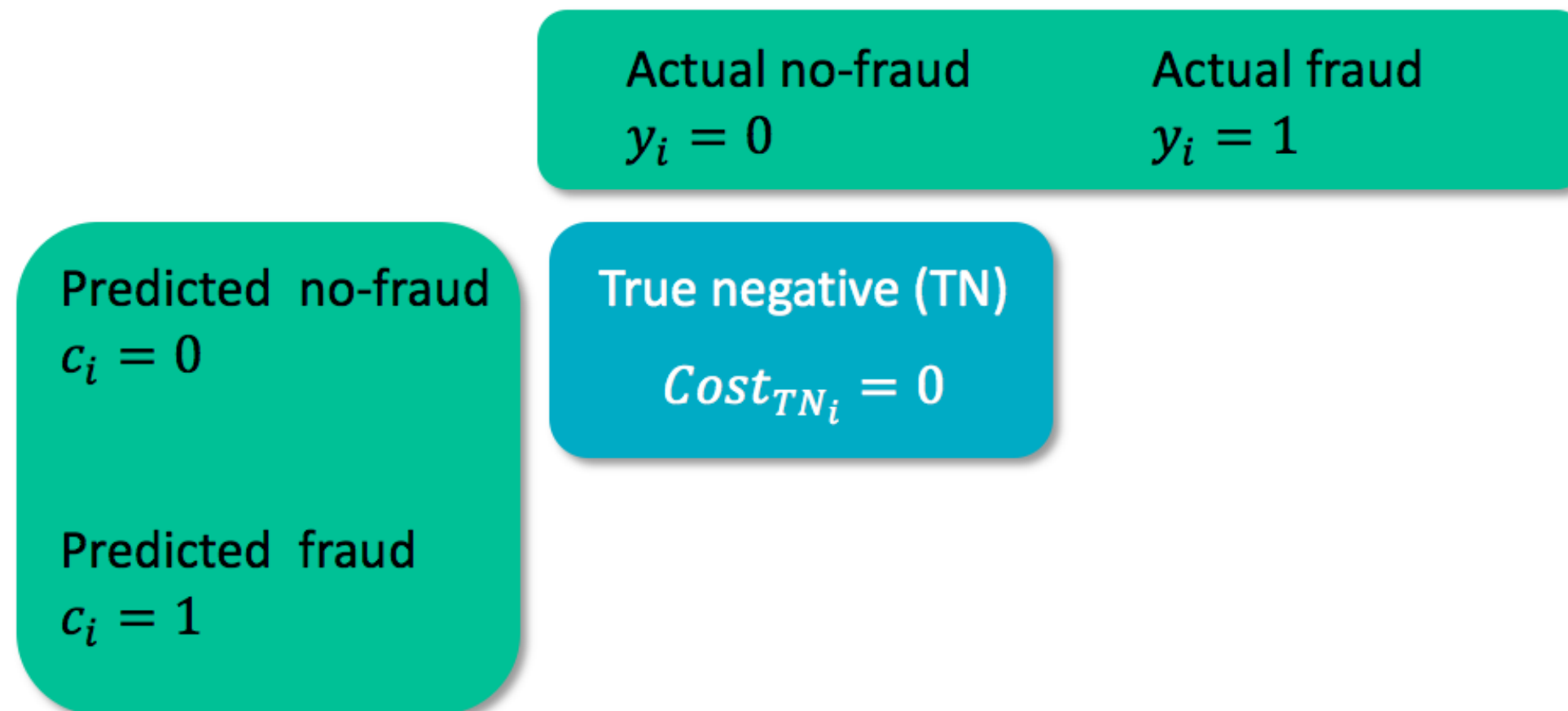



Cost of deploying a detection model

- Take into account the different **costs of fraud detection** during the evaluation of an algorithm
- Costs are associated with both **misclassification errors** (false positives & false negatives) and **correct classifications** (true positives & true negatives)

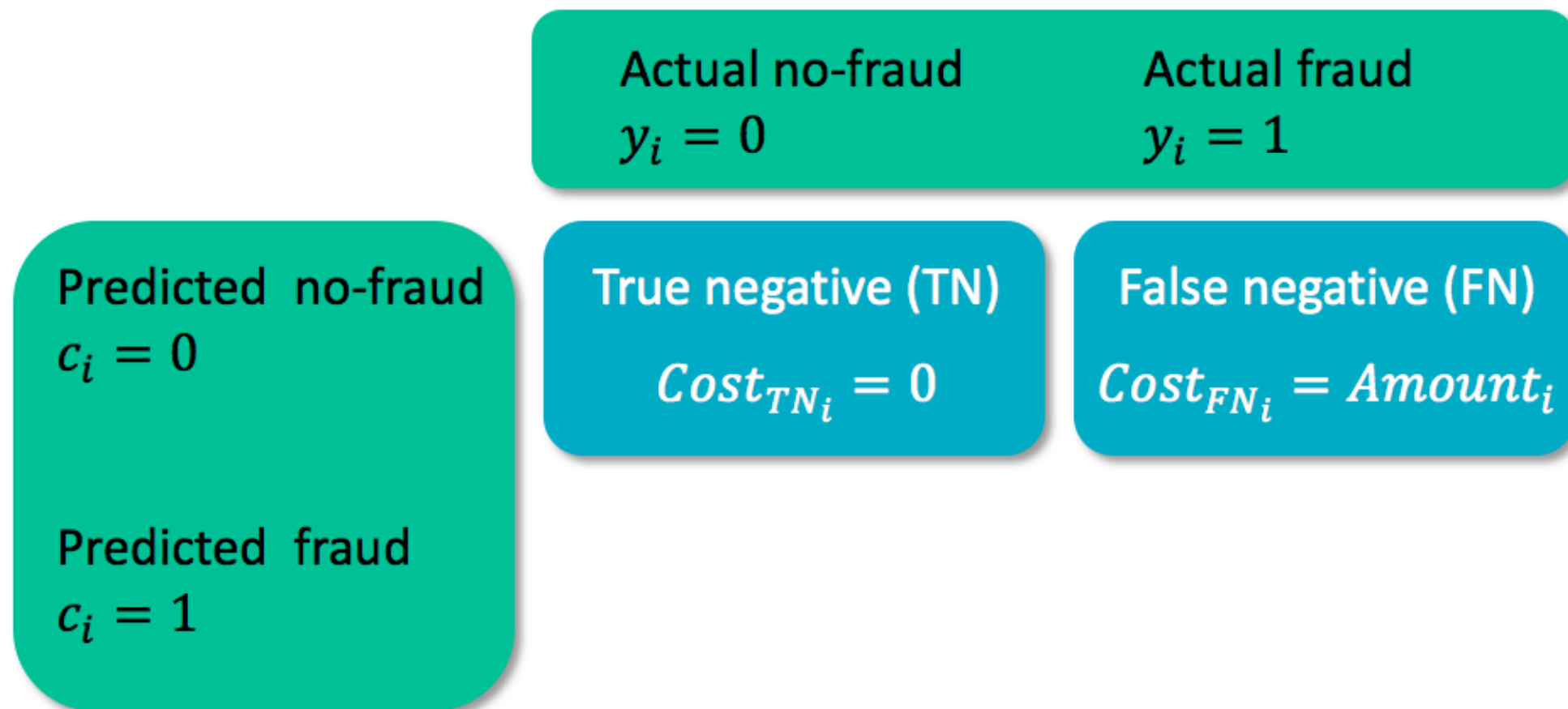


Cost matrix



- y_i = true class of case i
- c_i = predicted class for case i

Cost matrix



- y_i = true class of case i
- c_i = predicted class for case i



Cost matrix

	Actual no-fraud $y_i = 0$	Actual fraud $y_i = 1$
Predicted no-fraud $c_i = 0$	True negative (TN) $Cost_{TN_i} = 0$	False negative (FN) $Cost_{FN_i} = Amount_i$
Predicted fraud $c_i = 1$	False positive (FP) $Cost_{FP_i} = C_a$	

- C_a = cost for analyzing the case

Cost matrix

	Actual no-fraud $y_i = 0$	Actual fraud $y_i = 1$
Predicted no-fraud $c_i = 0$	True negative (TN) $Cost_{TN_i} = 0$	False negative (FN) $Cost_{FN_i} = Amount_i$
Predicted fraud $c_i = 1$	False positive (FP) $Cost_{FP_i} = C_a$	True positive (TP) $Cost_{TP_i} = C_a$

- C_a = cost for analyzing the case

Cost measure for a detection model

- Take into account the actual costs of each case:

$$Cost(model) = \sum_{i=1}^N y_i(1 - c_i)Amount_i + c_iC_a$$

- y_i = true class of case i
- c_i = predicted class for case i

```
cost_model = function(predicted.classes, true.classes, amounts, fixedcost) {  
  cost = sum(true.classes * (1 - predicted.classes) * amounts +  
             predicted.classes * fixedcost)  
  
  return(cost)  
}
```



True cost of fraud detection

```
# Total cost without using SMOTE:

cost_model(predicted_class1, test$Class, test$Amount, fixedcost = 10)

[1] 10061.8

# Total cost when using SMOTE:

cost_model(predicted_class2, test$Class, test$Amount, fixedcost = 10)

[1] 7431.93
```

- Losses decrease by 26%!



FRAUD DETECTION IN R

Let's practice!