

# Data Analytics Challenge - ASN-SMOTE

2023-07-02

Philipp von Lovenberg, Vincent Bläske, Dennis Götz

## Inhaltsverzeichnis

Einleitung

1. Data Exploration
2. Anwendung des SMOTE Algorithmus und ASN-SMOTE
3. Verwendete Classifier und Performancemaß
4. Cross-Validation
5. Undersampling
6. Hyperparameteroptimierung
7. Performance des ASN-SMOTE
8. Undersampling vor ASN-SMOTE

Ausblick

## Einleitung

Derzeit stellen unausgeglichene Datensätze eine erhebliche Herausforderung im Bereich des maschinellen Lernens und des Data Minings dar. Denn die herkömmlichen Klassifizierungsverfahren neigen in der Regel dazu, die Mehrheitsklasse zu bevorzugen, welche bei einem binären Klassifizierungsproblem oft ein Vielfaches an Instanzen gegenüber der Minderheitsklasse besitzt. Hierdurch wird der Klassifikator in seiner Vorhersagefähigkeit, die Minderheitsinstanzen korrekt zu erkennen, stark beeinträchtigt, kann jedoch trotzdem eine hohe Accuracy aufweisen. Die Minderheitsklasse kann dabei auch völlig unerkannt bleiben. Diese Thematik ist insbesondere bei der Erkennung betrügerischer Transaktionen in Banken, Kreditrisikobewertung oder Erkennung von Firewall-Eingriffen von hoher Bedeutung. Aufgrund dieser universellen Existenz von unausgewogenen Datensätzen wurden bereits viele Vorverarbeitungsmethoden vorgeschlagen, um die Imbalance zu bewältigen. Oversampling ist eine vielversprechende Technik für unausgewogene Datensätze, die neue Minderheitsinstanzen erzeugt, um den Datensatz auszugleichen. Unter den Oversampling-Methoden ist die Synthetic Minority-Oversampling-Technique (SMOTE) eine der bekanntesten Methoden, die künstliche Minoritätsinstanzen durch lineare Interpolation erzeugt. Eine jüngst veröffentlichte Adaption dessen liefert im Paper „ASN-SMOTE: a synthetic minority oversampling method with adaptive qualified synthesizer selection“ von Yi Xinkai et al. (2022) vielversprechende Ergebnisse, weshalb dieser Algorithmus im Folgenden auf dem Creditcard Datensatz angewendet wird. Weiterhin wird seine Performance mithilfe zweier geeigneter Classifier und den Ergebnissen des originalen SMOTE Algorithmus untersucht. Dafür wird nach der Data Exploration und Anwendung des SMOTE im 1. Kapitel die Funktionsweise des ASN-SMOTE sowie die Unterschiede zum ursprünglichen Algorithmus in Kapitel 2 erläutert. Anschließend werden die verwendeten Classifier und das Performancemaß in Kapitel 3 sowie die gewählte Cross-Validation im 4. Kapitel präsentiert. Darauf folgt in Kapitel 5 und 6 eine Methode des Undersamplings und die Durchführung der Hyperparameteroptimierung. Abschließend werden die Ergebnisse des ASN-SMOTE in Kapitel 6 analysiert, bevor ein Ausblick mit weiteren Ideen und Forschungsfragen folgt. Der Code des entsprechenden Abschnitts ist zur Nachvollziehbarkeit in der Programmiersprache R an jedes Kapitel angehängt.

## 1. Data Exploration

Der zu behandelnde Datensatz Creditdata beinhaltet europäische Kreditkartentransaktionen von September 2012 im CSV-Format und besitzt pro Transaktion 31 Features/Spalten, wobei die letzte binäre Spalte die Target darstellt. Diese definiert, ob ein Sample Fraud (1) oder kein Fraud (0) ist. Das Ziel hierbei ist es ein Machine Learning Modell zu trainieren, welcher Kreditkartenbetrug erkennen soll, um die Kunden vor unautorisierten Belastungen zu schützen. Die Daten sind ausschließlich numerisch und abgesehen von Time und Amount aus Datenschutzgründen mit PCA transformiert worden. Zu Beginn werden die notwendigen Packages geladen sowie die Working Directory in den Ordner gesetzt, worin sich der Creditdata Datensatz befindet und mit read.csv eingelesen. Anschließend wird eine NA-Analyse durchgeführt und einige Statistiken der einzelnen Spalten berechnet. Die Imbalance des Datensatzes wird deutlich, wenn wir die Anzahl der Minority Samples mit den 284.315 der Majority vergleichen. Denn es befinden sich lediglich 492 Fraud Samples in Creditdata, was einem Prozentsatz von 0,17% entspricht. Dies deutet auf eine extreme Imbalance zwischen Majority und Minority hin und macht eine Preprocessingmethode umso notwendiger. Aufgrund der hohen Imbalance wird ein 90 zu 10 Split der Daten in Trainings- und Testdatensatz durchgeführt, worauf eine Trennung der Features von der Target Variable folgt. Da das Feature Time als nicht aussagekräftig genug erscheint, wird dieses aus dem Datensatz entfernt und ebenfalls ein Scaling der Spalte Amount vorgenommen, da diese nicht mit PCA transformiert wurde. Dies geschieht in Trainings- und Testdatensatz separat, um den Mittelwert und die Standardabweichung des Testdatensatzes nicht mit in den Trainingsdatensatz aufzunehmen, was andernfalls zu Overfitting führen würde. Beim Korrelationsplot der Features mit der Target Variable fallen einige positive als auch negative Korrelation beim Feature Amount beispielsweise mit V2 und V7 auf. Die Target Value weist z.B. mit den Features V12, V15 und V17 lediglich einige niedrige negative Korrelation auf. Dies liegt insbesondere an der PCA Transformation, welche in Voraus durchgeführt wurde. Aus diesem Grund werden keine Features selektiert.

```
# Load necessary packages
library(tidyverse)
library(randomForest)
library(caTools)
library(smotefamily)
library(caret)
library(mlr)
library(tibble)
library(corrplot)
```

```
# Load the dataset
#setwd("C:/Users/Vincent BL/Desktop/DAC/")
setwd("C:/Users/Dennis/OneDrive/Dokumente/03_Master/05_Kurse/01_BA/04_DAC/")
ccdata <- read.csv("creditcard.csv")

# Look at the data
#View(ccdata)
#summary(ccdata)          # summary statistics of the data
colSums(is.na(ccdata))    # check for NA in the data
```

[illegible]

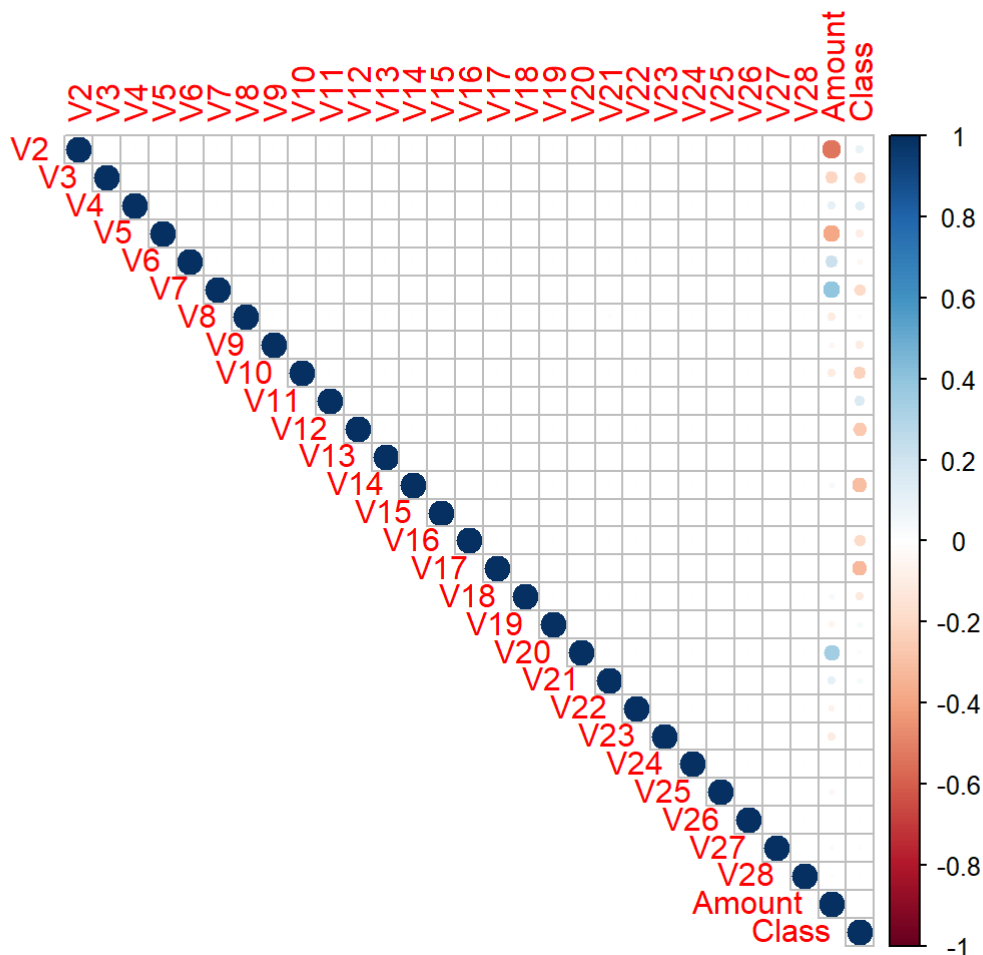
```
table(ccdata$Class)      # absolute amount of class membership
```

```
##  
##      0      1  
## 284315  492
```

```
### Preprocessing  
# Split into training/test set  
set.seed(123)  
split <- sample.split(ccdata$Class, SplitRatio = 0.9)  
train <- subset(ccdata, split == TRUE)  
test <- subset(ccdata, split == FALSE)  
table(train$Class)
```

```
##  
##      0      1  
## 255884  443
```

```
# Split features from the target value  
y_train <- train$Class  
X_train <- train[,-30]  
  
# Drop column 'Time' and scale column 'Amount' individually for train and test set  
train <- train[,-1] %>% mutate(Amount = scale(Amount))  
test <- test[,-1] %>% mutate(Amount = scale(Amount))  
  
### some data exploration (plot of time and amount for the 2 classes)  
# correlation  
corr_plot <- corrplot(cor(train[, -c(1)]), method = "circle", type = "upper")
```



## 2. Anwendung des SMOTE Algorithmus und ASN-SMOTE

### 2.1 Anwendung des SMOTE Algorithmus

Im Anschluss an die Data Exploration wird der in R bereits implementierte SMOTE Algorithmus aus dem Package `smotefamily` auf den Trainingsdatensatz angewendet, um eine Balance im Datensatz herzustellen. Nach der Generierung von 255.228 synthetischen Trainingsdaten beträgt der Anteil der Minority Samples im Train Set 49,97%. Beim Vergleich der Visualisierungen mit den Features V1 und V2 wird die Funktionsweise des SMOTE Algorithmus mit linearer Interpolation durch die geraden Linien, welche die synthetischen Daten ziehen, deutlich.

```
### Apply the original SMOTE Algorithmus to the train set
# Calculate number of synthetic samples for each minority instance
n_smote <- as.integer((255884 - 443)/443)

set.seed(1234)
smote_ <- smotefamily::SMOTE(X = X_train, target= y_train, K= 5, dup_size=n_smote)
train_smote <- (smote_$data)

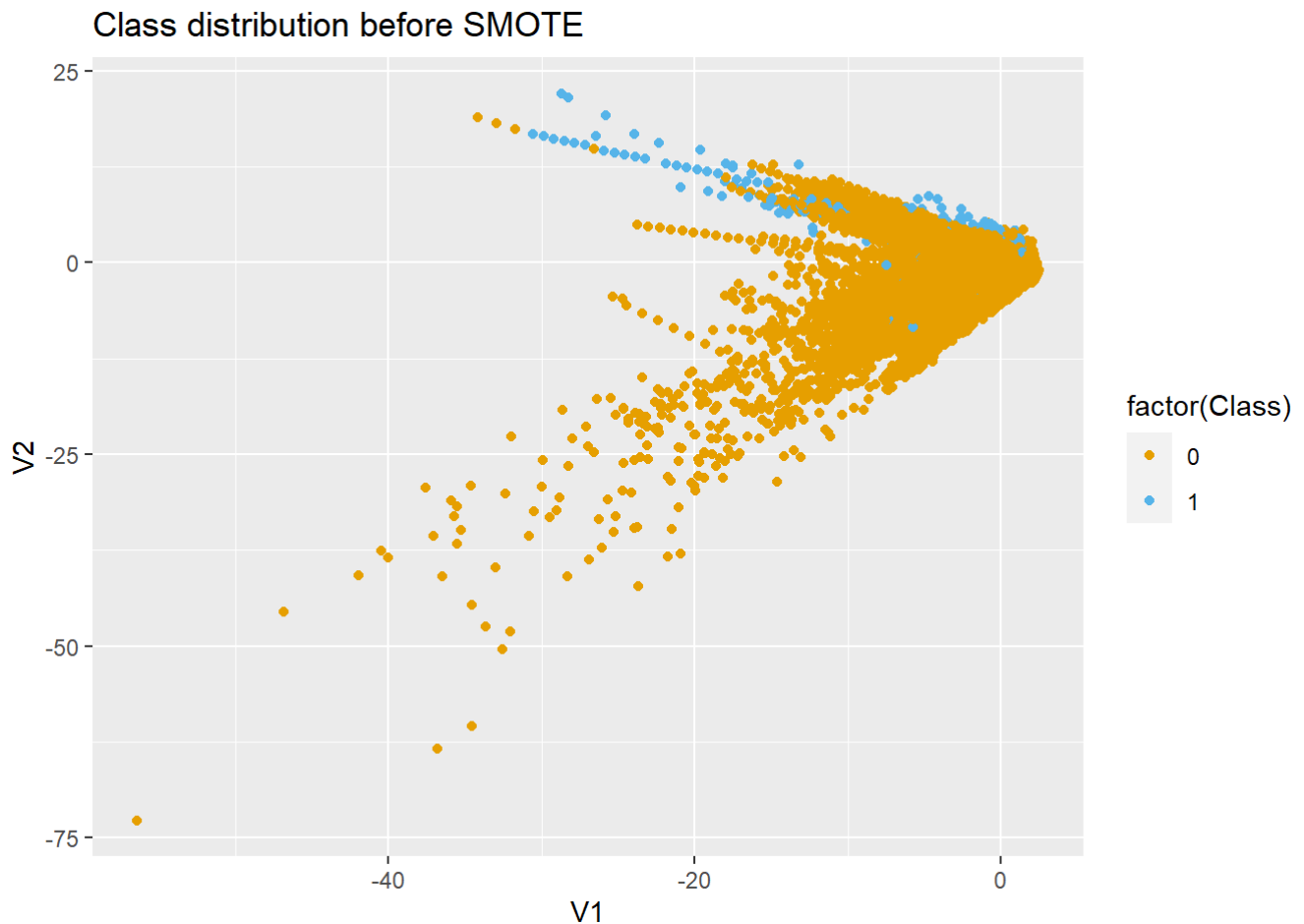
# View the new balance in the dataset
table(train_smote$Class)
```

```
##
##      0      1
## 255884 255611
```

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

```
##  
##           0           1  
## 0.5002669 0.4997331
```

```
# data visualization  
# Plot the first two features with the target value before SMOTE  
ggplot(train, aes(x = V1, y = V2, color = factor(Class))) + geom_point() + ggtitle("Class dist  
ribution before SMOTE") + scale_color_manual(values = c("#E69F00", "#56B4E9"))
```



```
# Plot the first two features with the target value before SMOTE  
ggplot(train_smote, aes(x = V1, y = V2, color = factor(class))) + geom_point() + ggtitle("Cla  
ss distribution after SMOTE")+ scale_color_manual(values = c("#E69F00", "#56B4E9"))
```

Class distribution after SMOTE



## 2.2 ASN-SMOTE

Allerdings können unsachgemäß erzeugte Minority Samples das Lernen des Classifiers auch beeinträchtigen und sich negativ auf ihn auswirken. Da im originalen SMOTE Algorithmus sowohl alle Minority Instanzen als auch alle k-nearest neighbors gleich behandelt werden und somit zur Generierung synthetischer Datenpunkte verwendet werden, kann dies zu einem Verschwimmen der Entscheidungsgrenze bzw. Überlappen der Feature Spaces führen. Durch die synthetischen Daten im Feature Space der Majority Class, wird es für ein Modell schwierig die beiden Klassen zu unterscheiden. Auf dieses Problem zielt der einfache und effektive Ansatz des ASN-SMOTE ab, welcher ebenfalls auf der k-Nearest-Neighbors und SMOTE-Technologie basiert. Dieser lässt sich in drei Algorithmen unterteilen, wobei der erste als Noise-Filtering bezeichnet wird. Hierbei wird für jede Minority Instanz die euklidische Distanz zu allen anderen Datenpunkten berechnet und lediglich der nächste Nachbar betrachtet. Sollte dieser zur Majority gehören, wird die gerade betrachtete Minority Instanz als Noise klassifiziert und im weiteren Verlauf nicht als Synthesizer für die Datengenerierung verwendet. Sollte der nächste Nachbar ebenfalls zur Minority gehören, ist die betrachtete Minority Instanz qualifiziert für die Generierung synthetischer Daten. Dies dient zur Filterung von Noise und Instanzen entlang der Entscheidungsgrenze. Im Anschluss folgt der Algorithmus zur adaptiven Nachbarschaftsauswahl, wobei für die k nächsten Nachbarn erneut die euklidische Distanz berechnet wird. Befindet sich darunter eine Majority Instanz, werden nur diejenigen als Nachbar für die Synthese genutzt, welche eine geringere Distanz als die nächste Majority Instanz aufweisen. Alle anderen Nachbarn werden als unqualifiziert markiert und nicht für die Datengenerierung mit der betrachteten Minority Instanz verwendet. Der dritte Algorithmus dient zur Erstellung der Samples, wobei anfangs die Anzahl der notwendigen neuen Instanzen für jedes Minority Sample bis zur optimalen Balance berechnet werden. Daraufhin folgt die Anwendung des SMOTE Algorithmus, welcher die synthetischen Daten mithilfe der linearen Interpolation erstellt. Dabei wird die euklidische Distanz zwischen der Synthesizer Minority Instanz und einem zufällig ausgewählten Nachbar der k nächsten Nachbarn (bei ASN: nur qualifizierten k nächsten Nachbarn) mit einer Zufallszahl zwischen Null und Eins multipliziert und zur Synthesizer Instanz addiert. Anschließend wird zu jeder erstellten Instanz die Target Variable (1) hinzugefügt und die synthetischen Daten an den Trainingsdatensatz angehängt. Die Vorteile des neuen Ansatzes sind

dabei die Effektivitätssteigerung durch das Filtern von Noise sowie die Feature Spaces daran zu hindern sich zu überlappen. Die Wirksamkeit dieses modernen ASN-SMOTE Algorithmus wurde mit 24 unausgewogenen Datensätzen im Originalpaper getestet und zeigte durchweg positive Ergebnisse im Vergleich mit anderen namhaften SMOTE Adaptionen. Die Imbalance der 24 Datensätze lag dabei zwischen 1,82% und 41,4%, weshalb die folgenden Fragen aufkommen: Funktioniert der ASN-SMOTE auch beim Creditdata Datensatz, welcher eine deutlich höhere Imbalance von 0,17% aufweist? Oder ist möglicherweise ein vorausgezogenes Undersampling notwendig? Bei der Anwendung des Algorithmus auf den gegebenen Datensatz, zeigt sich, dass 82 der lediglich 442 Minority Samples im Trainingsdatensatz als unqualifiziert klassifiziert werden, wodurch dem Classifier das Lernen noch mehr erschwert wird. Bei der Analyse von höheren Werten für den Hyperparameter k wird deutlich, dass nur die wenigsten Minority Instanzen 30 nächste Nachbarn aus der eigenen Klasse oder mehr besitzen und keine mehr als 72. Daher wird beim Tuning ein höheres Augenmerk auf niedrige Wert für k gelegt und dieser Bereich feiner unterteilt als hohe Werte für k. Aufgrund von Unstimmigkeiten im ASN-Paper wurde „n“ die Anzahl der zu generierenden Samples pro qualifizierter Minority Instanz sowohl als Inputparameter festgelegt als auch in der Funktion selbst „n\_opt“ berechnet. Durch Auskommentieren der Codezeile „n <- n\_opt“ kann n als Eingabewert verwendet werden und andernfalls wird jenes n gewählt, welches zur optimalen Balance im Datensatz führt. Beim Plot der Features V1 und V2 wird fällt anhand der zwei gekennzeichneten Punkte der Unterschied zum SMOTE Ansatz deutlich auf, da die vom ersten Algorithmus als Noise klassifizierte blaue Minority Instanz nicht für die Generierung synthetischer Daten verwendet wird. Zudem wählt der Algorithmus zur Nachbarauswahl den zweiten gekennzeichneten Punkt der Majority Class nicht als qualifizierten Nachbar aus. Das Verhalten des ASN ist bei Zweiterem aber möglicherweise ungünstig, da es sich hierbei um einen Ausreißer der Majority handeln könnte, welcher im Feature Space der Minority liegt.

```

# ASN-SMOTE function
asn_smote <- function(train, n, k) {

  # Split the train set into features (T in the Pseudo Code) and target value
  train_feat <- train[,1:29]
  train_target <- train$Class

  # Create a matrix with the features and split the train set into majority and minority
  train_feat_matrix <- as.matrix(train_feat)
  train_Majority <- train[train_target == 0,]
  train_Minority <- train[train_target == 1,]

  # Select only the features of the minority train set (P in the Pseudo code)
  train_Minority_feat <- train_Minority[,1:29]

  # Calculate the distance of each minority instance to all samples of the train set
  dis_matrix <- proxy::dist(train_Minority_feat, train_feat)

  # Create a List with indices of the k-nearest minority neighbors of all minority
  # instances (majority neighbors marked as NaN)
  index_knn <- list()

  for (i in 1:nrow(train_Minority_feat)) {
    index_knn[[rownames(dis_matrix)[i]]] <- order(dis_matrix[i,])[2:(k+1)]
    for (j in 1:k) {
      if (train_target[index_knn[[i]][j]] == 0 ) {
        index_knn[[i]][j] <- NaN
      }
    }
  }

  print("Distance matrix calculated and nearest neighbors defined.")
  print("-----")

  # Algorithm 1: Filter Noise
  # Drop minority instances with a majority (NaN) as nearest neighbor
  Mu <- vector()

  for (i in length(index_knn):1) {
    if (is.nan(index_knn[[i]][1])) {
      Mu[i] <- names(index_knn[i])
      index_knn <- index_knn[-i]
    }
  }

  print(paste0("Number of qualified minority instances: ", length(index_knn),
    " of ", nrow(train_Minority)))
  print("Algorithm 1 successfully completed.")
  print("-----")

  # Algorithm 2: Adaptive neighbor instances selection
  # Keep only the neighbors that are closer than the nearest majority (NaN) instance

```



```

for (i in 1:length(index_knn)) {
  for (j in 1:k) {
    if (is.nan(index_knn[[i]][j])) {
      index_knn[[i]] <- index_knn[[i]][1:(j-1)]
      break
    }
  }
}

print(paste0("Mean qualified nearest neighbors: ",
            round(sum(lengths(index_knn))/length(index_knn), 2), " of ", k))
print(paste0("Maximum qualified nearest neighbors: ",
            max(sapply(index_knn, length)), " of ", k))
hist_qn <- hist(sapply(index_knn, length), plot=FALSE)
print("Algorithm 2 successfully completed.")
print("-----")

# Algorithm 3: Procedure of ASN-SMOTE (Create new synthetic minority samples)
# Add to the feature values of each qualified minority instance the difference of the
# minority sample and one random selected neighbor of their qualified neighbors
# multiplied with a random number between 0 and 1 for n times.

# Calculate the amount of synthetic minority samples for each qualified minority sample
# that the train set is balanced
n_opt <- as.integer((nrow(train_Majority) - nrow(train_Minority))/length(index_knn))
print(paste0("optimal n = ", n_opt))

#if you assign 'n_opt' to 'n', then 'n' is not a inputparameter anymore
# always the perfect balance will be generated in the dataset
#n <- n_opt

synthetic <- list()
for(i in names(index_knn)) {
  for(j in seq_len(n)) {
    random_n <- sample(seq_along(index_knn[[i]]), 1)
    dif <- train_feat_matrix[index_knn[[i]][random_n],] - train_feat_matrix[i,]
    randomNum <- runif(1)
    synthetic_instance <- train_feat_matrix[i,] + randomNum * dif
    synthetic[[length(synthetic) + 1]] <- synthetic_instance
  }
}

print(paste0("Number of generated synthetic minority samples: ", length(synthetic)))
print("Algorithm 3 successfully completed.")
print("-----")

# Assign "Class" label = 1 to the synthtic points
synthetic_df <- do.call(rbind, synthetic)
synthetic_df <- as.data.frame(synthetic_df)
synthetic_labels <- rep(1, length(synthetic))
synthetic_df$Class <- synthetic_labels

# Combine original train set with synthetic set

```

```

asn_train <- rbind(train, synthetic_df)
print("The ASN-SMOTE was applied to the data.")
print("The new training dataset is saved as 'asn_train'.")
print("-----")

# View the new balance of the dataset
print("New balance of 'creditcard' dataset:")
return (table(asn_train$Class))
}

```

```

# Execute ASN-SMOTE function (optimal balanced with n=707)
asn_smote(train, n=10, k=5)

```

```

## [1] "Distance matrix calculated and nearest neighbors defined."
## [1] "-----"
## [1] "Number of qualified minority instances: 361 of 443"
## [1] "Algorithm 1 successfully completed."
## [1] "-----"
## [1] "Mean qualified nearest neighbors: 4.34 of 5"
## [1] "Maximum qualified nearest neighbors: 5 of 5"
## [1] "Algorithm 2 successfully completed."
## [1] "-----"
## [1] "optimal n = 707"
## [1] "Number of generated synthetic minority samples: 3610"
## [1] "Algorithm 3 successfully completed."
## [1] "-----"
## [1] "The ASN-SMOTE was applied to the data."
## [1] "The new training dataset is saved as 'asn_train'."
## [1] "-----"
## [1] "New balance of 'creditcard' dataset:"

```

```

##
##      0      1
## 255884  4053

```

```

# Execute ASN-SMOTE function with high 'k' (very high k doesn't make sense!)
asn_smote(train, n=10, k=100)

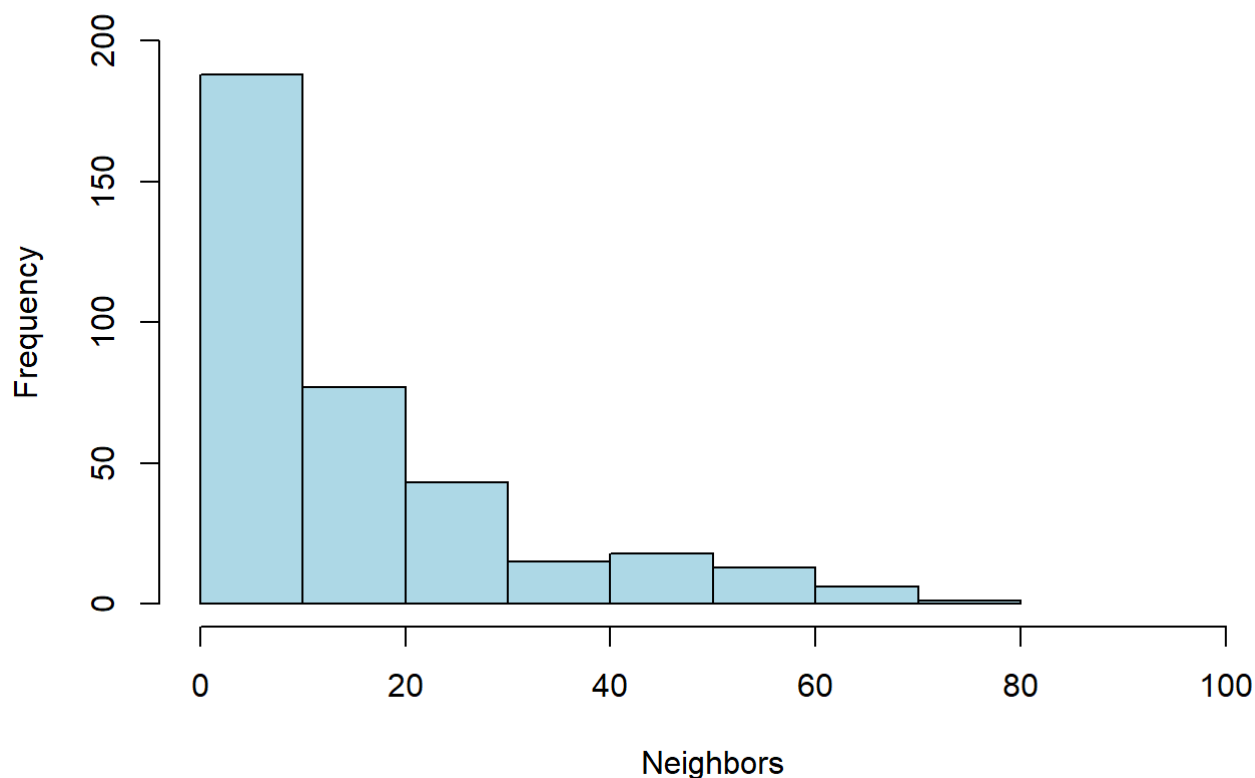
```

```
## [1] "Distance matrix calculated and nearest neighbors defined."
## [1] "-----"
## [1] "Number of qulaified minority instances: 361 of 443"
## [1] "Algorithm 1 successfully completed."
## [1] "-----"
## [1] "Mean qualified nearest neighbors: 15.68 of 100"
## [1] "Maximum qualified nearest neighbors: 72 of 100"
## [1] "Algorithm 2 successfully completed."
## [1] "-----"
## [1] "optimal n = 707"
## [1] "Number of generated synthetic minority samples: 3610"
## [1] "Algorithm 3 successfully completed."
## [1] "-----"
## [1] "The ASN-SMOTE was applied to the data."
## [1] "The new training dataset is saved as 'asn_train'."
## [1] "-----"
## [1] "New balance of 'creditcard' dataset:"
```

```
##
##      0      1
## 255884 4053
```

```
plot(hist_qn, xlab = "Neighbors", ylab = "Frequency", xlim = c(0, 100), ylim = c(0, 200),
     col = "lightblue", main = "Histogram of qualified nearest neighbors")
```

**Histogram of qualified nearest neighbors**



```
# data visualization after ASN-SMOTE
#ggplot(asn_train, aes(x = V1, y = V2, color = factor(class))) + geom_point() + ggtitle("Class distribution after ASN-SMOTE")+ scale_color_manual(values = c("#E69F00", "#56B4E9"))
```

## 3. Verwendete Classifier und Performancemaß

### 3.1 Verwendete Classifier

Um für das Testdatenset Vorhersagen treffen zu können, müssen wir erst ein Modell aufstellen und dieses tunen. Hierbei entscheiden wir uns für die Machine Learning Modelle der Logistischen Regression und des Random Forest. Die Logistische Regression eignet sich beispielsweise gut für die Schätzung der Wahrscheinlichkeiten von binären oder kategorialen abhängigen Variablen. Der grundlegende Ansatz der logistischen Regression besteht darin, dass eine Sigmoid-Funktion auf eine lineare Kombination der Features angewendet wird. Die Sigmoid-Funktion transformiert den linearen Ausdruck in einen Wert zwischen 0 und 1, der als Wahrscheinlichkeit interpretiert werden kann. Zudem findet eine Regularisierung statt, welche mit dem Hyperparameter C festgelegt wird. Dieser ist auch der einzige zu tunende Parameter im Modell, wodurch eine Cross-Validation mit geringerem Rechenleistung durchgeführt werden kann. Jedoch könnte die vom Modell angenommene lineare Beziehung zwischen den Features einen Nachteil darstellen. Im Bezug auf den Random Forest ist zu wissen, dass er zu den Ensemble-Lernverfahren gehört und für Klassifikations- und Regressionsaufgaben verwendet wird. Er besteht aus einer Kombination von Entscheidungsbäumen, die unabhängig voneinander trainiert werden. Dabei wird jeder Baum auf einer zufälligen Stichprobe der Trainingsdaten trainiert und betrachtet auch nur eine zufällige Teilmenge der verfügbaren Features. Dies hilft dem Modell, um Overfitting zu vermeiden und gut verallgemeinern zu können. Bei einer Klassifikationsaufgabe ist die Vorhersage üblicherweise eine Mehrheitsentscheidung der Bäume. Die Vorteile dieser Methode sind, dass der Random Forest problemlos mit großen Datenmengen arbeiten kann und nicht dazu neigt die Trainingsdaten auswendig zu lernen. Zudem schneidet er in der Praxis oft besser ab als andere Classifier. Beim Tuning sind jedoch deutlich mehr Hyperparameter zu optimieren, wodurch ein großer Rechenaufwand notwendig ist und das Modell ist meist schwieriger zu interpretieren als einfachere Verfahren, wie z.B. die Logistische Regression.

### 3.2 Performancemaß

Die Leistung der Algorithmen des maschinellen Lernens wird in der Regel anhand einer Konfusionsmatrix bewertet und die Vorhersagegenauigkeit definiert als  $Accuracy = (TP + TN) / (TP + FP + TN + FN)$ . Dabei ist die Fehlerrate  $1 - Accuracy$ . Dies gilt jedoch nur im Zusammenhang mit ausgeglichenen Datensätzen und gleichen Fehlerkosten. Denn beim betrachteten Datensatz würde sogar ein Modell, welches immer die Majority prognostiziert, eine Accuracy von über 99% erreichen. Daher verwenden wir beim Creditdata Datensatz mit ungleichen Fehlerkosten, die Receiver Operating Characteristic Curve (ROC-Curve) und die Area under the Curve (AUC), um die Performance der Logistischen Regression und des Random Forests zu bestimmen. Hierbei wird auf der X-Achse die False-Positive-Rate  $\%FP = FP / (TN + FP)$  und auf der Y-Achse die True-Positive-Rate  $\%TP = TP / (TP + FN)$  dargestellt. Der ideale Punkt auf der ROC-Kurve wäre (0,100), d. h. alle positiven Samples werden richtig klassifiziert und keine negativen Beispiele werden falsch als positiv klassifiziert. Die Linie  $y = x$  stellt das Szenario einer zufälligen Schätzung der Klasse dar. Zudem ist die Fläche unter der ROC Kurve (AUC) ein nützliches Maß für die Leistung des Klassifikators, da es unabhängig vom gewählten Entscheidungskriterium und den vorherigen Wahrscheinlichkeiten ist.

```
#Code
```

## 4. Cross-Validation

*#Code*

## 5. Undersampling

*#Code*

## 6. Hyperparameteroptimierung

*#Code*

## 7. Performance des ASN-SMOTE

*#Code*

## 8. Undersampling vor ASN-SMOTE

*#Code*

## Ausblick

Aufgabengebiete Philipp von Lovenberg: Durcharbeiten SMOTE-Paper, Mitarbeit bei der Erstellung der Präsentationsfolien, Mitarbeit bei der Recherche nach modernen SMOTE Adaptionen, Erstellung der Cross-Validation, Hyperparameter tuning von SMOTE-Algorithmus, Durchführen des Undersamplings, Mitarbeit bei der Erstellung des R Markdown Dokuments.

Aufgabengebiete Vincent Bläske: Durcharbeiten SMOTE-Paper, Mitarbeit bei der Erstellung der Präsentationsfolien, Mitarbeit bei der Recherche nach modernen SMOTE Adaptionen, Implementierung des SMOTE-Algorithmus, Durcharbeiten des ASN-SMOTE Papers, Mitarbeit bei der R Implementierung des ASN-SMOTE, Auswahl und Implementierung der Classifier, Hyperparameter tuning der Classifier, Plotten, Performanceauswertung, Mitarbeit bei der Erstellung des R Markdown Dokuments.

Aufgabengebiete Dennis Götz: Durcharbeiten SMOTE-Paper, Mitarbeit bei der Erstellung der Präsentationsfolien, Mitarbeit bei der Recherche nach modernen SMOTE Adaptionen, Durcharbeiten des ASN-SMOTE Papers und der dazugehörigen Python Implementierung, Mitarbeit bei der R Implementierung des ASN-SMOTE und des Modells der logistischen Regression, Mitarbeit bei der Erstellung des R Markdown Dokuments.