Machine Learning

Support Vector Machines

Hauptseminar

am Fachgebiet Wirtschaftsinformatik für Industriebetriebe
der Fakultät für Wirtschaftswissenschaften und Medien

Verfasser:	Tobias Rummelsberger (Matr.-Nr. 50542)
	Benjamin Wörrlein (Matr-Nr. 56995)
Betreuer:	Dr. Sören Bergmann
Hochschullehrer:	Univ.-Prof. Dr.- Ing. S. Straßburger

Diese Seminararbeit wurde am 21.01.2018 in der Fakultät für Wirtschaftswissenschaften und Medien der Technischen Universität Ilmenau eingereicht.

Zusammenfassung

Mit der zunehmenden Digitalisierung von Produktions- und Logistiksystemen sind immer mehr Daten über diese Prozesse verfügbar. In Verbindung mit der Entwicklung und Einführung von Machine Learning Methoden können mit den gesammelten Stellgrößen, Leistungskennzahlen mit geeigneten Modellen vorhergesagt werden. Im Rahmen dieser Arbeit wird die Methode Support Vector Machine (SVM) vorgestellt. Dafür wird eine Einführung in die mathematischen Grundlagen gegeben und verschiedene Arten der SVM vorgestellt. Des Weiteren werden die verwendeten Werkzeuge im Sinne von Software und Paketen beschrieben. Anschließend wird der Prozess der Datenvorverarbeitung und Modellerstellung anhand eines Beispieldatensatzes durchgeführt. Die Güte der Modelle wird darauf folgend verglichen und bewertet. Abschließend wird eine kritische Würdigung durchgeführt.

Diese Arbeit wurde am 21.01.2018 in der Fakultät für Wirtschaftswissenschaften und Medien der Technischen Universität Ilmenau eingereicht.

Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich diese Masterarbeit selbstständig ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe. Alle den benutzten Quellen wörtlich oder sinngemäß entnommenen Stellen sind als solche einzeln kenntlich gemacht.

Diese Arbeit ist bislang keiner anderen Prüfungsbehörde vorgelegt worden und auch nicht veröffentlicht worden.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Ilmenau, 21.01.2018 _____

Tobias Rummelsberger

Ilmenau, 21.01.2018 _____

Benjamin Wörrlein

Inhalt

ABKÜRZUNGSVERZEICHNIS ..... **VI**

TABELLENVERZEICHNIS ..... **VII**

ABBILDUNGSVERZEICHNIS ..... **VIII**

FORMELVERZEICHNIS ..... **IX**

1 EINLEITUNG **1**

2 GRUNDLAGEN **2**

2.1 KNOWLEDGE DISCOVERY IN DATABASES 2
2.2 MACHINE LEARNING 2
2.3 IDENTIFIKATIONSVERFAHREN 3

3 SUPPORT VECTOR MACHINES **4**

3.1 HISTORISCHE ENTWICKLUNG UND EINORDNUNG 4
3.2 GRUNDLAGEN 4
3.3 SUPPORT VECTOR REGRESSION 9

4 WERKZEUGE **9**

4.1 PROGRAMMIERSPRACHE 9
4.2 ENTWICKLUNGSUMGEBUNG 9
4.3 PAKETE 10

5 DATENSATZ **12**

5.1 EXPLORATORY DATA ANALYSIS 12
5.2 VORBEREITUNG DES DATENSATZES 13

6 ANWENDUNG **15**

6.1 BERECHNUNG DER SVM UND PARAMETRIERUNG 15
6.2 EVALUATION 17

7 KRITISCHE WÜRDIGUNG **21**

LITERATURVERZEICHNIS **22**

ANHANG A: HERLEITUNG LINEARER SVM ..... **X**

ANHANG B: HERLEITUNG SOFT MARGIN CLASSIFIER ..... **XII**

ANHANG C: PRODUKTIONSDATENSATZ - INTERVALLE ..... **XIII**

ANHANG D: SYSTEMÜBERSICHT ..... **XIV**

ANHANG E: KLASSEN DER LEISTUNGSKENNZAHLEN ..... **XV**

ANHANG F: EVALUATION VON SVC UND SVR ..... **XVI**

ANHANG G: LIESMICH-DATEI ..... **XXVII**

ANHANG H: CODE ..... **XXIX**

Abkürzungsverzeichnis

CM	Confusion Matrix
EDA	Exploratory Data Analysis
GPA	Generalized Portrait Algorithm
KDD	Knowledge Discovery in Databases
ML	Machine Learning
NAE	normalized absolute error/ Normalisierte absolute Abweichung
SVC	Support Vector Classifier
SVM	Support Vector Machines/ Stützvektormachine
SVR	Support Vector Regression

Tabellenverzeichnis

Tabelle 1: Wichtige Identifikationsverfahren im Überblick [BV2008] 3
Tabelle 2: Datenerfassung (Stellgrößen und Leistungskennzahlen) 12
Tabelle 3: Parametrierung der SVC 16
Tabelle 4: Parametrierung der SVR 17
Tabelle 5: Kreuztabelle für Buffer2Max 17
Tabelle 6: Genauigkeit der SVC 18
Tabelle 7: mittlere und maximale prozentuale Abweichung der SVR 20
Tabelle 8: links: Kreuztabelle der Regression, rechts: Kreutabelle der Klassifikation 21

Abbildungsverzeichnis

Abbildung 1: KDD-Prozess nach Usama Fayyad [FPS1996], eigene Darstellung 2
Abbildung 2: lineare SVM, Hard Margin [eigene Darstellung] 7
Abbildung 3: lineare SVM, Soft Margin [eigene Darstellung] 8
Abbildung 4: Entwicklungsumgebung R Studio [eigene Darstellung] 9
Abbildung 5: Histogramm normierte absolute Abweichung Throughput 19
Abbildung 6: Evaluation der SVR 20

Formelverzeichnis

Formel 1: Definition des Begriffes Machine Learning [Ha2009] 2

Formel 2: Abhängigkeit des abhängigen Merkmals Y von unabhänigen Merkmalen [BV2008] 3

Formel 3: Das grundlegende Modell der Diskriminanzanalyse nach [BV2008] 3

Formel 4: Matrixdarstellung der multiplen Regression [BV2008] 4

Formel 5: Hyperebene im p-dimensionalen Raum, nach [JWHT2013] 5

Formel 6: (1) der Bereich oberhalb der Hyperebene, (2) Bereich unterhalb der Hyperebene 5

Formel 7: Datenmatrix X [JWHT2013], Seite 339 6

Formel 8: Eigenschaften der Hyperebene nach [JWHT2013], Seite 340 6

Formel 9: Formeln für verschiedene typische Kernel nach [HK2006] 8

Formel 10: Anzahl Modelle One against One [Me2017] 11

Formel 11: Min-Max-Normalisierung nach [HK2006] 14

Formel 12: Normierte absolute Abweichung nach [HK2006] 18

1 Einleitung

Problemstellung

Durch die wachsende Rechenleistung von Computern und die Weiterentwicklung von Algorithmen hat die Verwendung von maschinellem Lernen in den vergangenen Jahren stark zugenommen. Dieser Trend ist ebenfalls in Produktions- und Logistiksystemen zu erkennen, bei denen aus gesammelten Daten zukünftige Kennzahlen vorhergesagt werden. Im Rahmen dieser Arbeit wird auf die Grundlagen von Support Vector Machines (SVM) eingegangen und anschließend der Anwendungsprozess inklusive der Datenvorbereitung anhand von einem Datensatz aus Produktion und Logistik aufgezeigt und diskutiert.

Zielsetzung

Ziel dieser Arbeit ist es die Methode Support Vector Machine, welche dem Machine Learning zugeordnet wird, vorzustellen und diese im Rahmen eines vorgegebenen Datensatzes zu erproben. Mit den Daten, welche aus einem Produktions- und Logistiksystem stammen, wird ein prädiktives Vorhersagemodell erstellt, trainiert und optimiert. Mit dem Hintergrund der getroffenen Annahmen und der Vorverarbeitung des Datensatzes wird anschließend eine kritische Würdigung verfasst und die Eignung der Methode der Support Vector Machine auf den gegebenen Datensatz diskutiert.

Methodik und Vorgehensweise der Arbeit

Als methodische Grundlage dieser Arbeit wurde der KDD-Prozess nach Usama Fayyad [FPS1996] gewählt, welcher neben der Erstellung des Modells auch die Bildung von Vorwissen, die Datenreinigung, Auswahl einer passenden Methode und eines passenden Modells, sowie die Evaluation und Interpretation umfasst.
Im Rahmen des KDD-Prozesses wird schließlich ein Softwaresystem zur Berechnung eines prädiktiven Modells entwickelt, welches auf Klassifikation und Regression mittels Support Vector Machines basiert.

Aufbau der Arbeit

Im ersten Teil der Arbeit wird der Prozess der Knowledge Discovery in Databases erläutert und die Entwicklung eines geeigneten Modells in diesen eingeordnet. Anschließend wird eine Einführung in die mathematischen Grundlagen der Support Vector Machines gegeben. Außerdem wird im Rahmen von beispielhaften Anwendungsfällen auf einige Sonderfälle eingegangen. Darauf folgend wird eine geeignete Software zur Entwicklung eines SVM-Modells ausgewählt. Des Weiteren werden die Annahmen zur Bearbeitung genannt und erläutert. Anschließend wird auf die Erstellung des SVM-Modells eingegangen und die Ergebnisse diskutiert und die Ergebnisse sowie die Eignung von Support Vector Machines im gegebenen Kontext kritisch gewürdigt.

2 Grundlagen

2.1 Knowledge Discovery in Databases

Als methodische Grundlage für diese Arbeit wird der KDD-Prozess nach U. Fayyad gewählt. Der von U. Fayyad 1989 geprägte Begriff definiert die Schritte der Einordnung der Daten in den Kontext, der Datenvorverarbeitung (Selektion und Reinigung) und einer sachgemäßen Evaluation und Interpretation als essenziell für die Anwendung von Data Mining [FPS1996]. Da eine nicht korrekte Ausführung der Vor- und Nachbereitung der Daten die Gefahr birgt falsche Muster und Modelle zu extrahieren, wird auf diese Schritte im Rahmen dieser Arbeit ebenfalls eingegangen.

Abbildung 1: KDD-Prozess nach Usama Fayyad [FPS1996], eigene Darstellung

Der KDD-Prozess läuft iterativ ab, sodass nach jedem Schritt gegebenenfalls zu einem der vorherigen Schritte zurückgesprungen werden kann. Ein Iterationsschritt kann beispielsweise nötig sein, wenn bei der Modellbildung auffällt, dass die Eingangsdaten für das Modell nicht angemessen vorverarbeitet wurden oder das Modell nicht korrekt ausgewählt wurde. Durch diese Schritte wird eine kontinuierliche Optimierung des Modells durchgeführt. Die für die Wissensgewinnung erforderlichen Informationen werden durch Algorithmen des Machine Learning gewonnen und bilden den zentralen Baustein des KDD-Prozesses [Kü1999] und [Al2003].

2.2 Machine Learning

Das Ziel von Machine Learning ist deskriptive und prädiktive Modelle zur Beschreibung und Vorhersage von Daten auf Basis von einem Datensatz zu erstellen. Durch das Trainieren an Beispielen erstellt ein künstliches System verallgemeinerte Modelle.

Formel 1: Definition des Begriffes Machine Learning [Ha2009]

- $Datenmenge\ X$
- $Stichprobe\ S, mit\ S \subset X$
- $Zielfunktion\ f: X \rightarrow \{true, false\}$
- $Trainingssatz\ D, mit\ D = \{(true, false) \mid x \in S\ und\ y = f(x)\}$

$$Berechnung\ einer\ Funktion\ \hat{f}: X \rightarrow \{true, false\}\ mit\ Trainingssatz\ D, sodass$$
$$\hat{f}(x) \cong f(x), \qquad \forall\ x \in X$$

Der Trainingsdatensatz beschreibt eine Untermenge aus dem gesamten Datensatz. Dieser wird verwendet, um das Modell für alle Datenelemente des Datensatzes zu verallgemeinern. Somit können Muster und Regelmäßigkeiten in Datensätzen erkannt werden. Maschinelles Lernen findet in vielen Bereichen wie Predicitve Maintenance (vorausschauende Wartung) oder Sprach- und Texterkennung Anwendung und ist eng mit Knowledge Discovery in Databases und Data Mining verknüpft. Neben der von Han (*Formel 1*) be-

schriebenen Definition des Begriffes Machine Learning, wird Machine Learning auch für die Regression von Daten eingesetzt.

2.3 Identifikationsverfahren

Zur Untersuchung der Zugehörigkeit von Objekten zu Klassen werden Identifikationsverfahren eingesetzt. Ein abhängiges Merkmal *Y* wird mit unabhängigen Merkmalen $X_{1,} \dots X_m$ erklärt, sodass

Formel 2: Abhängigkeit des abhängigen Merkmals Y von unabhänigen Merkmalen [BV2008]

$$Y = f(X_1, \dots X_m)$$

gilt. Da *Y* als abhängige Variable von mehreren unabhängigen Merkmale *X* beschrieben wird, ist dieses Verfahren der multivariaten Statistik zuzuordnen. Diese kausalen Abhängigkeiten können sowohl linear als auch nicht-linear sein, wobei im Rahmen der Betrachtung von SVMs auf beide Fälle eingegangen wird. Die Identifikationsverfahren lassen sich nach Skalenniveau der abhängigen und unabhängigen Merkmale unterteilen.

Unabhängige Merkmale / Abhängige Merkmale	**quantitativ**	**nominal**
quantitativ	Regressionsanalyse	Varianzanalyse
nominal	Diskriminanzanalyse	Kontingenzanalyse

Tabelle 1: Wichtige Identifikationsverfahren im Überblick [BV2008]

Bei Objekten, deren Klassenzugehörigkeit unbekannt ist, kann auf Basis einer mit den Identifikationsverfahren erstellten Prognose die Klassenzugehörigkeit abgeschätzt werden. Im Rahmen dieser Arbeit wird auf die Diskriminanzanalyse und die Regressionsanalyse eingegangen. Die (multiple) Regressionsanalyse wird angewandt, wenn sowohl das abhängige als auch die unabhängigen Merkmale quantitativ sind. Die Diskriminanzanalyse wird herangezogen, wenn das abhängige Merkmal nominal ist und die unabhängigen Merkmale quantitativ sind. Also wenn beispielsweise auf Basis eines Produktmixes mit den jeweiligen Anteilen der Produkte berechnet werden soll, ob die Maschinengeschwindigkeit über oder unter dem Durchschnitt liegt.

Diskriminanzanalyse

Die Diskriminanzanalyse dient der Zuordnung von Objekten zu nominalen Klassen in Abhängigkeit von quantitativen Merkmalen. Das grundlegende Modell der Diskriminanzanalyse wird folgendermaßen dargestellt:

Formel 3: Das grundlegende Modell der Diskriminanzanalyse nach [BV2008]

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \cong \begin{pmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_m \end{pmatrix} = X \cdot \vec{w}$$

Regressionsanalyse

Die Regressionsanalyse beschreibt die Abhängigkeit eines quantitativen Merkmales *Y* von quantitativen Merkmalen $X_{1,\dots}$ X_m. Das Grundmodell mit m unabhängigen Merkmalen und n Beobachtungswerten lässt sich folgendermaßen beschreiben:

Formel 4: Matrixdarstellung der multiplen Regression [BV2008]

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_m \end{pmatrix} + \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix} = X \cdot \vec{w} + \vec{u}$$

Hier stellt *Y* die Regressanten dar, und die unabhängigen Variablen $X_{1,\dots}$ X_m werden Regressoren genannt, U fungiert als Störvariable. Die Gewichtung *w* der unabhängigen Variablen *X* erfolgt so, dass die Varianz innerhalb der Klassen möglichst klein und zwischen den Klassen möglichst groß wird.

3 Support Vector Machines

3.1 Historische Entwicklung und Einordnung

Das Konzept der Support Vector Machine im Sinne der Support Vector Classifier basiert auf der Idee von Ronald A. Fisher eine Funktion zu bestimmen, die Klassen in einer Datenmenge möglichst gut voneinander trennt und somit Muster erkennt. Die Klassentrennung soll durch eine Funktion erfolgen, die zu beiden Klassen einen möglichst großen Abstand hat [Fi1936]. Im Folgenden entwickelte Frank Rosenblatt einen ersten linearen Klassifikationsalgorithmus, welcher Perzeptron genannt wurde und den einfachsten Fall eines neuralen Netzes beschreibt [Ro1958]. Ab 1963 folgte dann die Weiterentwicklung durch Vladimir N. Vapnik mit der Einführung des Generalized Portrait Algorithm, welcher als nichtlineare Generalisierung in SVMs implementiert wird [VL1963]. Ein weiterer wichtiger Schritte für die Entwicklung von SVMs war die Einführung der Kernelfunktionen, um ebenfalls nicht lineare separierbare Daten mit einem geringen Rechenaufwand klassifizieren zu können [ABR1964]. Neben weiteren Entwicklungen war die Publikation von Vladimir N. Vapnik 1992 auf der COLT-Konferenz von besonderer Bedeutung, da diese bereits nah an der Form heutiger SVMs ist. Darauf folgend veröffentliche Vladimir N. Vapnik im Jahr 1995 noch ein Erweiterung der SVM um den Soft Margin Classifier [VC1995] und die Erweiterung zur Support Vector Regression [Vl1995].

3.2 Grundlagen

Begriffliche Grundlagen

Die grundlegende Idee von Support Vector Machine bzw. Support Vector Classifier ist, eine Datenmenge mittels eines Unterraumes (Hyperebene[1]) in zwei Klassen zu unterteilen. Dieser Unterraum hat im n-dimensionalen Raum die Dimension n-1. Diese Hyperebene wird ausschließlich von den Stützvektoren (Support Vector) aufgespannt, welche die Datenpunkte sind, die sich am nächsten an der Hyperebene befinden. Mit dieser Methode kann neben der Klassifikation ebenso eine Regression von Daten durchgeführt werden.

[1] In der engl. Literatur „Hyperplane“

Somit sind SVM die Verallgemeinerung von Breiter-Rand-Klassifikatoren (engl. Maximum Margin Classifier) für nicht-lineare sowie mehrdimensionale Anwendungen. Im Folgenden werden die mathematischen Grundlagen anhand des zweidimensionalen Falles erklärt.

Mathematische Grundlagen

Zunächst wird eine Hyperebene definiert. Die Hyperebene ist ein p-1-dimensionaler Unterraum, welcher die Klassen voneinander trennt. Im p-dimensionalen Fall wird die Hyperebene durch folgende Gleichung bestimmt:

Formel 5: Hyperebene im p-dimensionalen Raum, nach [JWHT2013]

$$\sum_{i=1}^{p} w_i \cdot X_i + b = w_1 X_1 + w_2 X_2 + \cdots + w_p X_p + b = 0$$

Entsprechend wird mit *Formel 6: (1) der Bereich oberhalb der Hyperebene, (2) Bereich unterhalb der Hyperebene* beschrieben.

Formel 6: (1) der Bereich oberhalb der Hyperebene, (2) Bereich unterhalb der Hyperebene

(1) $\sum_{i=1}^{p} w_i \cdot X_i + b = w_1 X_1 + w_2 X_2 + \cdots + w_p X_p + b > 0$

(2) $\sum_{i=1}^{p} w \cdot X_i + b = w_1 X_1 + w_2 X_2 + \cdots + w_p X_p + b < 0$

Somit wird der Datenraum in zwei Klassen eingeteilt und die Erstellung einer beliebigen Hyperebene vorgenommen. Ziel der SVM ist allerdings die optimale Hyperebene zu berechnen. Diese wird vorgenommen, indem um die Hyperebene ein Rand gebildet wird, der anschließend maximiert wird. Innerhalb des Randes liegen keine Datenpunkte. Die Stützvektoren oberhalb und unterhalb der Hyperebene erfüllen die folgenden Gleichungen:

(1) $\sum_{i=1}^{p} w_i \cdot X_i + b = w_1 X_1 + w_2 X_2 + \cdots + w_p X_p + b = +1$

(2) $\sum_{i=1}^{p} w_i \cdot X_i + b = w_1 X_1 + w_2 X_2 + \cdots + w_p X_p + b = -1$

Zunächst werden die Gleichungen mittels Verwendung des Skalarproduktes vereinfacht zu:

(1) $\sum_{i=1}^{p} w_i \cdot X_i + b = \langle \vec{w}, \vec{x} \rangle = +1$

(2) $\sum_{i=1}^{p} w_i \cdot X_i + b = \langle \vec{w}, \vec{x} \rangle = -1$

Durch Einsetzen von zwei Stützvektoren $\overrightarrow{x_1}$ und $\overrightarrow{x_2}$ und Subtraktion der genannten Gleichungen ergibt sich der Abstand zwischen zwei Stützvektoren zu.

$$\langle \vec{w}, (\overrightarrow{x_1} - \overrightarrow{x_2}) \rangle = 2$$

Durch Normieren von $\vec{w}$ erhält man den Abstand zwischen den beiden Stützvektoren in die Richtung des zur Hyperebene orthogonalen Vektors $\vec{w}$:

$$\langle \frac{\vec{w}}{\|w\|}, (\overrightarrow{x_1} - \overrightarrow{x_2}) \rangle = \frac{2}{\|w\|}$$

Daher wird eine Maximierung des Randes, dessen Breite durch den oben beschriebenen Abstand zwischen den Stützvektoren berechnet, durch Minimieren von $\|\vec{w}\|$ erreicht. Die

weiteren mathematischen Schritte zur Minimierung von $\|\vec{w}\|$ sind in *Anhang A: Herleitung linearer SVM* zu finden.

Lineare Klassifikation mit SVM

Mit einem Datensatz **X**, welcher aus n Trainingselementen in einem p-dimensionalen Raum besteht und die Beobachtungen den beiden Klassen $y \in \{-1, 1\}$ zugeordnet werden wird ein System zur Klassifikation entwickelt. Die Datenmatrix X und der Ergebnisvektor y hat folgendes Format:

Formel 7: Datenmatrix X [JWHT2013], Seite 339

$$x_1 = \begin{pmatrix} x_{11} \\ \dots \\ x_{1p} \end{pmatrix}, \dots, x_n = \begin{pmatrix} x_{n1} \\ \dots \\ x_{np} \end{pmatrix} \quad \vec{y} = \begin{pmatrix} y_1 \\ \dots \\ y_n \end{pmatrix}$$

Das Ziel ist nun die Klassen möglichst gut voneinander zu trennen, somit hat die separierende Hyperebene die Eigenschaften:

Formel 8: Eigenschaften der Hyperebene nach [JWHT2013], Seite 340

(1) $f(x) = \langle w, x \rangle + b = \sum_{i=1}^{n} w_i x_i + b = x_1 + w_2 x_2 + \cdots + w_p x_p > 0$ für $y_i = +1$

(2) $f(x) = \langle w, x \rangle + b = \sum_{i=1}^{n} w_i x_i + b = x_1 + w_2 x_2 + \cdots + w_p x_p < 0$ für $y_i = -1$

und somit gilt:

$$y_i(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p) > 0$$

Mit $f(x)$ kann somit bestimmt werden, zu welcher Klasse ein Datenelement mit dem Vektor $\overrightarrow{x^*}$ zuzuordnen ist. Neben der Zuordnung zu einer der Seiten kann ebenfalls aus dem Betrag der Formel darauf geschlossen werden, wie weit das Datenelement von der Hyperebene entfernt ist.

Im Folgenden wird die optimale Berechnung der Hyperplane beschrieben, so dass diese zwei Klassen möglichst gut voneinander trennt. Es gibt eine unendlich große Anzahl an nicht optimalen Hyperebenen, da diese normalerweise immer ein bisschen verschoben werden können. Das Ziel der optimalen Hyperebene ist zwischen den beiden Klassen eine möglichst große Distanz[2] zwischen den Klassen zu berechnen. Dies wird durch Maximierung des Randes erreicht. Die mathematischen Grundlagen sind im *Anhang A: Herleitung linearer SVM* zu finden.

[2] Engl.: maximal margin/ optimal seperating hyperplane

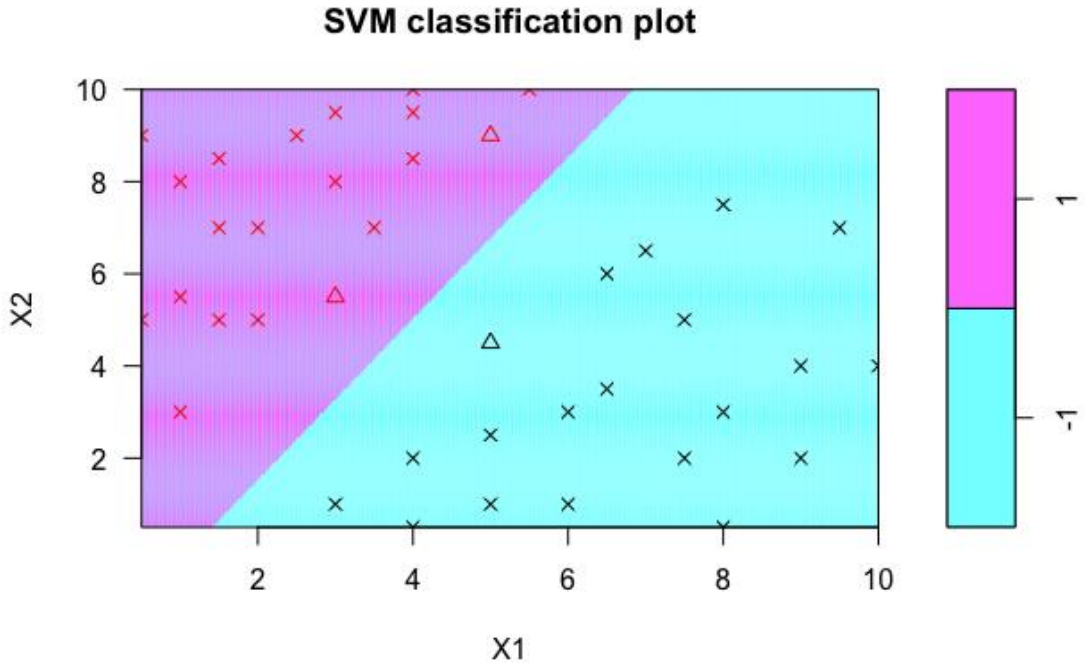


Abbildung 2: lineare SVM, Hard Margin [eigene Darstellung]

Wie in *Abbildung 2: lineare SVM, Hard Margin [eigene Darstellung]* zu erkennen ist, sind die drei der Hyperebene am nächsten gelegenen Datenelemente die Stützvektoren (Dreieck Symbol). Die weiteren Datenpunkte sind außerhalb des Randes, welcher zwischen den Stützvektoren gebildet wird.

Soft Margin Classifier

Es lassen sich nicht alle Datensätze linear trennen, daher wird für solche Datensätze der Ansatz der Soft Margin Klassifikation genutzt. Bei diesem Vorgehen wird zugelassen, dass Datenelemente innerhalb des Randes bzw. auf der falschen Ebene der Hyperebene liegen. Somit wird einer extremen Veränderung der Hyperebene bei Hinzufügen eines neuen Datenelementes, welches als Stützvektor fungiert, entgegengewirkt. Die Nutzung von Soft Margin Classifier beugt ebenfalls der Überanpassung[3] von SVMs an Trainingsdatensätze vor.

Die Anzahl der Datenelemente innerhalb des Randes und auf der falschen Seite der Hyperebene können durch den Kostenparameter und das Fehlergewicht γ gesteuert werden. Je höher das Fehlergewicht gewählt wird, desto breiter ist der Rand und somit werden mehr Datenelemente als Stützvektoren in die Modellbildung einbezogen.

[3] engl.: Overfitting

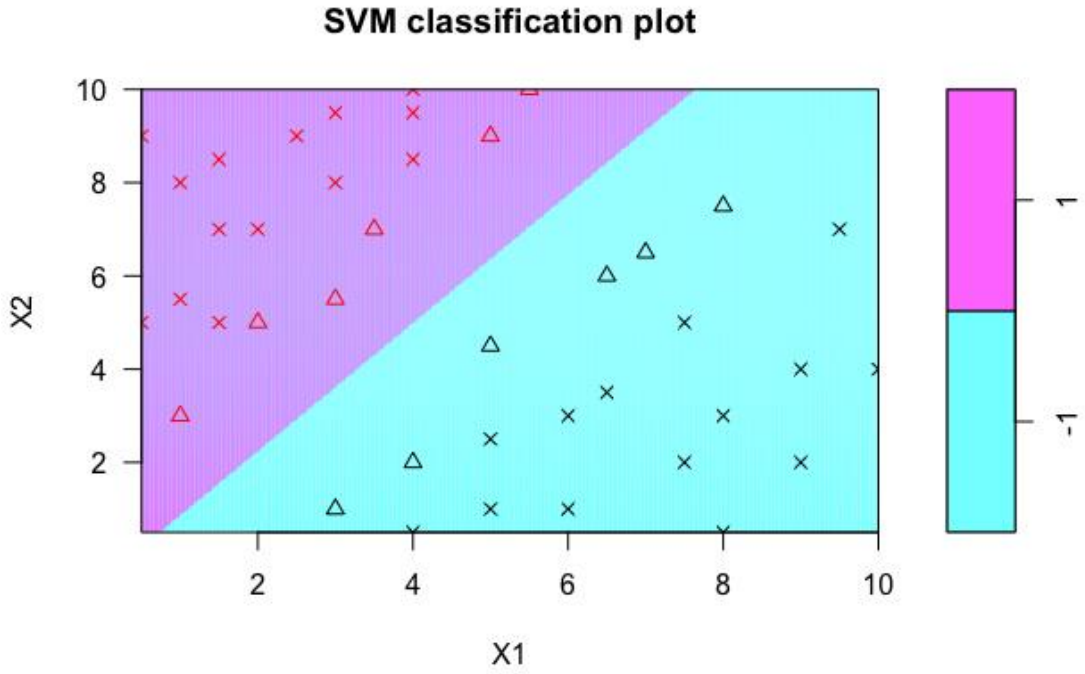


Abbildung 3: lineare SVM, Soft Margin [eigene Darstellung]

Wie sich in *Abbildung 3: lineare SVM, Soft Margin [eigene Darstellung]* erkennen lässt, steigt die Anzahl der Stützvektoren (Dreieck) durch den Einsatz von Soft Margin Classifier. Die Stützvektoren befinden sich auf dem Rand oder innerhalb bzw. auf der falschen Seite des Randes.

Nicht-lineare Klassifikation – Kernel-Trick

Neben der Soft Margin Klassifikation gibt es auch noch die Möglichkeit den Kernel-Trick zu nutzen, um eine bessere Performance von SVM zu erreichen. Typischerweise werden ein Polynom vom Grad d, die radiale Basis oder ein neuronales Netzwerk genutzt. Die genannten Funktionen sind typischerweise:

Formel 9: Formeln für verschiedene typische Kernel nach [HK2006]

Polynom vom Grad d: $k(x_i, x_j) = (c + \langle x_i, x_j \rangle)^d$

Radiale Basis: $k(x_i, x_j) = exp\left(-\frac{\|x_i - x_j\|^2}{c}\right)$

Neuronales Netzwerk: $k(x_i, x_j) = \tanh(\kappa \langle x_i, x_j \rangle + \theta)$

Die Grundidee des Kernel-Tricks ist es, den Datensatz in einen höherdimensionalen Raum zu transformieren und dort linear zu trennen. Dies erfordert im Normalfall einen hohen Rechenaufwand. Im Fall der SVM ist aber der Vorteil, dass nur die Skalarprodukte der Datenelemente berechnet werden müssen. Durch diesen Sachverhalt befindet sich der Rechenaufwand in einem akzeptablen Rahmen. Bei der Berechnung von SVM ist sorgfältig auf die Auswahl des Kernels zu achten, da dieser das Ergebnis maßgeblich beeinflusst.

3.3 Support Vector Regression

Bei numerischen (kontinuierlichen) Daten ist die Durchführung einer Regression sinnvoll. Hier wird die Hyperebene so angepasst, dass möglichst viele Datenpunkte in der Nähe und somit innerhalb des Randes liegen. Die Klassifikation kann durch Diskretisierung der Daten sowohl für kontinuierliche als auch diskrete Werte eingesetzt werden. Die Regression hingegen wird nur bei kontinuierlichen Daten eingesetzt. Bei der Regression findet der Kernel-Trick ebenfalls Anwendung.

4 Werkzeuge

4.1 Programmiersprache

Als Programmiersprache wird R verwendet, welche größtenteils als statistische Programmiersprache, aber auch für die Manipulation und Visualisierung von Daten verwendet wird. Die Programmiersprache wurde von Ross Ithaka und Robert Gentleman an der Universität von Auckland als freie Software entwickelt. Die Sprache orientiert sich syntaktisch an der kommerziellen Sprache S, welche ebenfalls für statistische Zwecke genutzt wird. Die Entwicklung begann in den frühen 1990er Jahren und wurde als General Public License Version 2 im Juni 1995 zur Verfügung gestellt. Im Februar 2000 wurde schließlich die erste Version von R (1.0.0) veröffentlicht und seitdem stetig weiterentwickelt. Die Programmiersprache kann unter www.cran.r-project.org heruntergeladen werden. Für diese Arbeit wird die Version 3.4.2 verwendet.

4.2 Entwicklungsumgebung

Als Entwicklungsumgebung wird die Software R Studio im Rahmen dieser Arbeit genutzt und empfohlen, welche als Open Source Lizenz auf *www.rstudio.com* kostenlos heruntergeladen werden kann. Diese Entwicklungsumgebung birgt den Vorteil, dass neben dem Editor ebenfalls die Konsole und der Workspace bzw. die Historie und der Dateiexplorer bzw. Grafiken oder Pakete angezeigt werden. Außerdem ist ebenfalls die Hilfe der R-Hilfeseiten integriert.

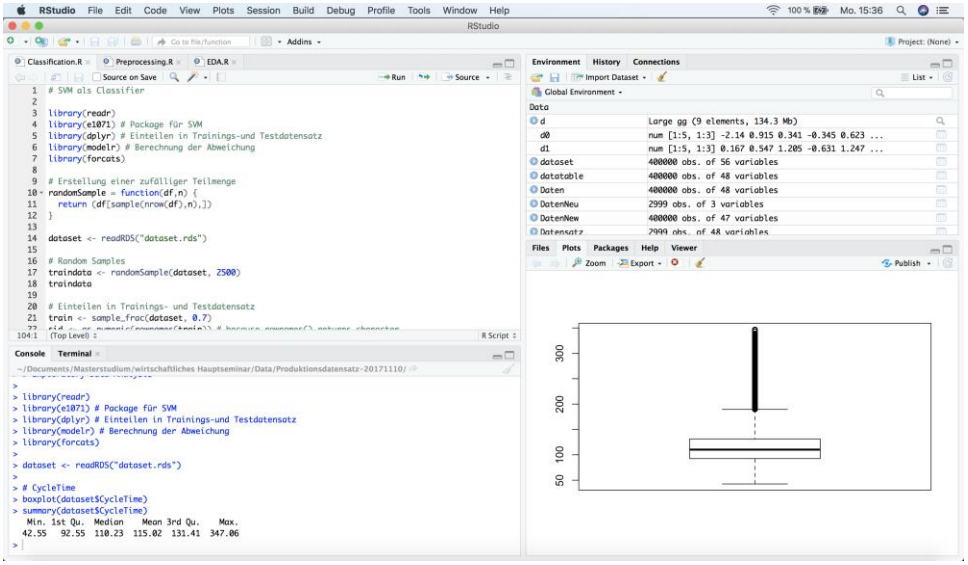


Abbildung 4: Entwicklungsumgebung R Studio [eigene Darstellung]

4.3 Pakete

Um den Bearbeitungsprozess dieser Arbeit nachvollziehen zu können, sollten folgende Pakete in R mittels folgendem Befehl heruntergeladen werden:
Install.packages(„package“)
Für die Datenmanipulation, Datenanalyse und Berechnung der SVMs wurden folgende Pakete verwendet:

- binr
- discretization
- e1071
- dplyr
- modelr

Genaue Beschreibungen der Pakete lassen sich auf *https://cran.r-project.org* finden.
Im Folgenden wird auf die Library e1071 bzw. LIBSVM näher eingegangen und diese weiter erläutert.

Das Paket e1071/ LIBSVM

Für die Berechnung der SVMs wird das Paket e1071 verwendet, welches auf LIBSVM basiert. Dieses ist auf *https://www.csie.ntu.edu.tw/~cjlin/libsvm/* für eine große Anzahl an gängigen Programmiersprachen erhätlich. LIBSVM wurde an der National University in Taiwan entwickelt und das Paket e1071 auf dessen Basis an der Wirtschaftsuniversität Wien für die Programmiersprache R entwickelt.
Im Paket e1071 stehen die folgenden Arten von SVMs zur Verfügung:

- C-Classification
- nu-Classification
- one-Classification
- eps-Regression
- nu-Regression

Die Standardwerte hier sind entweder C-Classification, wenn die abhängige Variable ein Faktor ist, ansonsten eps-Regression. Wenn keine abhängige Variable, also Ausgangsgröße angegeben wird, wird die One-Classification durchgeführt. Für die vollständigen Standardeinstellungen und Parameter wird auf [Me2017] verwiesen. Im Folgenden wird kurz auf die verschiedenen Arten der SVMs eingegangen.

C-Classification

Die C-Classification wird als Standardwert verwendet, wenn eine Klassifikation durchgeführt wird. Bei diesem Typ wird die SVM mit den Parametern Cost und gamma beeinflusst. Dadurch lässt sich das Kostenbudget und der Kernel-Parameter für den Kernel radiale Basis anpassen.

ν -Classification

Dieser Typ der SVM erlaubt über den Parameter ν eine Einflussnahme auf den Anteil an Stützvektoren an den Trainingsdatenpunkten.

One-Classification

Dieser Typ der SVM wird zur Neuigkeitsdetektion eingesetzt. Das Modell versucht die Verteilung zu approximieren und anhand dessen Ausreißer und Neuigkeiten zu entdecken.

ε -Regression

Bei der Regression liegen die Datenpunkte nicht außerhalb der beiden Ränder, sondern ein möglichst großer Anteil soll innerhalb liegen. Die Ränder werden hier unter gegebenen Randbedingungen maximiert.

ν -Regression

Analog zur ν-Klassifikation wird das Modell zur ν-Regression gebildet.

Multiklassen Klassifikation

Eigentlich sind SVMs nur für die binäre Klassifikation geeignet, durch den Einsatz von geeigneten Bewertungsschemata können SVMs aber auch für Datensätze eingesetzt werden, deren Elemente in mehrere Klassen eingeteilt werden. Somit wird das Multiklassenproblem in mehrere binäre Klassifikationen unterteilt.

Als Ansätze für Multiklassen SVMs gibt es *One versus All* und *One against One*. Bei erstgenannten werden die Elemente einer Klasse gegen die Elemente aller anderen Klassen getestet. Ein nicht klassifiziertes Element wird der Klasse mit dem höchsten *Confidence-Score* zugeordnet.

Bei *One against One* werden die Klassen paarweise, also einzeln gegeneinander getestet. Ein Element wird schließlich der Klasse zugeordnet, zu der es am häufigsten zugeordnet wurde. Das für diese Arbeit verwendete Software-Paket e1071 unterstützt die *One against One* Technik. Der Rechenaufwand beträgt bei *k* Klassen:

Formel 10: Anzahl Modelle One against One [Me2017]

$$k - 1 \cdot \frac{k}{2}$$

Anwendung

Im Folgenden wird die Anwendung des Software-Paketes in R beschrieben. Eine SVM wird erstellt, indem die Funktion *svm()* aufgerufen wird. Als Eingangsdaten müssen die Formel und der Datensatz eingegeben werden. Die Formel beschreibt die abhängige Variable *Y* – bei Klassifikation also die Klassenzugehörigkeit – in Abhängigkeit von den unabhängigen Variablen *X1* und *X2*. Des Weiteren wird der Datensatz eingegeben, auf dessen Basis die SVM trainiert werden soll. Als Parameter können unter anderem noch der Kernel und das Fehlerbudget bestimmt werden, in Abhängigkeit von Art der SVM werden noch weitere Parameter berücksichtigt.

```
example.svm <- svm(Y~X1+X2, data = example.df, kernel = “radial“, cost = 0.3, gamma = 0.01)
```

Die Funktion svm() gibt folgende Komponenten zurück: Die Matrix der gefundenen Stützvektoren, sowie deren Label im Klassifikationsmodus, außerdem noch deren Label

aus dem Input-Datensatz. Falls die Kreuzvalidierung aktiviert ist, werden ggf. noch weitere Werte zurückgegeben.
Für die Optimierung der Parameter wird die Funktion *tune()* verwendet. Hier werden die Parameter ähnlich wie bei der Erstellung einer SVM übergeben. Da diese Funktion allerdings für verschiedene Algorithmen einsetzbar ist, muss zuvor noch die Methode bestimmt werden. Im Fall dieser Arbeit also *svm*. Im Argument *ranges* werden die zu optimierenden Parameter sowie die Bereiche, innerhalb deren das Optimum gesucht werden soll, übergeben. Als Ausgabewert wird die Anzahl an Stützvektoren und die optimalen Werte zurückgegeben. Im Beispiel wird innerhalb des Kostenbudgets im Intervall $(0, 100)$ für alle ganzen Zahlen und für den Gammawert $[0, 1)$ in 0,1-Schritten nach der optimalen Lösung gesucht. Mit der Funktion *plot()* kann das Ergebnis als Raster ebenfalls visualisiert werden.

```
example.svm.tune <- tune(svm, Y~X1+X2, data = example.df, kernel = "radial", ranges = (cost = 1:100, gamma = c(0, 1, 0.1))
```

5 Datensatz

5.1 Exploratory Data Analysis

Der gegebene Datensatz besteht aus 37 Stellgrößen und 10 Leistungskennzahlen mit jeweils 400.000 Beobachtungswerten, also 18.800.000 Datenelementen. Die Daten beschreiben dabei folgende Attribute des zu modellierenden Systems:

Tabelle 2: Datenerfassung (Stellgrößen und Leistungskennzahlen)

	Attributname	Beschreibung
Stellgrößen	LoadingVolume	Auftragslast
	Buffer1, Buffer2	Kapazität der Puffer
	NumberOfCarriers	Anzahl Werkstückträger
	CarrierSpeedLoaded	Geschwindigkeit im beladenen Zustand
	CarrierSpeedEmpty	Geschwindigkeit bei Leerfahrt
	ProductX [1-27]	%-Anteil von Produkt X im Auftragsmix
	MachineXSpeed [1-4]	Effizienzfaktor für Maschine X
Leistungskennzahlen	Throughput	Gesamtausbringungsmenge
	Buffer1Mean, Buffer2Mean	Durchschnittliche Belegung der Puffer
	Buffer1Max, Buffer2Max	Maximal beobachtete Belegung der Puffer
	CycleTime	Durchschnittliche Durchlaufzeit eines Auftrages
	MachineXUtilization [1-4]	Durchschnittliche Auslastung von Maschine X

Eine Systemübersicht ist *Anhang D: Systemübersicht* zu entnehmen. Hier lässt sich auch erkennen, dass Buffer1 für alle Maschinen verwendet wird und Buffer2 nur für Maschine 4 genutzt wird.
Bei Betrachtung der Leistungskennzahlen Buffer1Max und Buffer2Max fällt auf, dass hier nur die diskreten Werte $\{1, 2\}$ bzw. $\{1, 2, 3, 4, 5, 6\}$ angenommen werden. Demnach kann hier ohne Diskretisierung eine Klassifikation und eine Multiklassen-Klassifikation der Daten vorgenommen werden. Die weiteren Leistungskennzahlen nehmen kontinuierliche Werte an und müssen daher vor einer Klassifikation erst diskretisiert werden.

5.2 Vorbereitung des Datensatzes

Zunächst wird der gegebene Datensatz eingelesen und als R-Objekt gespeichert, um die Lesezeit und die Schreibzeit zu senken. Des Weiteren werden Fehler in der Spaltenbeschriftung korrigiert.
Im nächsten Schritt wird ein Dataframe mit den Leistungskennzahlen und ein Dataframe mit den Stellgrößen erstellt. Dies ist nötig, da die Vorverarbeitung dieser Daten unterschiedlich abläuft. Die Stellgrößen werden normalisiert und die Leistungskennzahlen werden diskretisiert. Durch die Normalisierung der Stellgrößen wird erreicht, dass alle Variablen den gleichen Einfluss auf die Modellbildung haben und keine Variable auf Grund von vergleichsweise kleinen oder großen Zahlen eine größere oder geringere Berücksichtigung findet. Die Diskretisierung der Leistungskennzahlen ist nötig, damit eine SVM zur Klassifikation erstellt werden kann.
Die Attribute Buffer1Max und Buffer2Max beschreiben die maximale Belegung der Puffer und nehmen daher diskrete Werte an. Hier ist es sinnvoll eine Klassifikation durchzuführen, diese kann allerdings nur auf Faktoren angewendet werden. Daher wird hier der Datentyp von numerisch auf Faktor geändert:

```
dataset$Buffer1Max <- factor(dataset$Buffer1Max)
dataset$Buffer1Max <- factor(dataset$Buffer1Max)
```

Des Weiteren werden die abhängigen Variablen, also die Stellgrößen, normalisiert.

Normalisierung

Neben einer Diskretisierung der Leistungskennzahlen werden die Stellgrößen normalisiert, damit die Daten mit höheren absoluten Werten keinen größeren Einfluss in der Berechnung haben. Für die Normalisierung können verschiedene Methoden angewendet werden:

- Min-Max-Normalisierung
- Z-Score-Normalisierung
- Normalisierung mittels dezimaler Skalierung

Im Rahmen dieser Arbeit wurde die Min-Max-Normalisierung gewählt, welche die Daten innerhalb des Intervalls $[0{,}0; 1{,}0]$ normalisiert. Die Werte werden nach folgender Formel berechnet:

Formel 11: Min-Max-Normalisierung nach [HK2006]

$$V_i' = \frac{v_i - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$$

Diese Formel wird für das Intervall [0,0; 1,0] in R folgendermaßen beschrieben:

```
normalize <- function(x) { return ((x-min(x) / max(x) - min(x)))}
```

Diskretisierung

Die Diskretisierung von Daten, das heißt Umwandlung kontinuierlicher Daten in diskrete Teilmengen, wird durchgeführt, um die Datenmenge für die Klassifikation vorzubereiten. Dabei werden die kontinuierlichen Daten in Intervalle unterteilt. Die Intervallbezeichnungen repräsentieren die ursprüngliche Datenmenge.

Für die Diskretisierung können verschiedene Algorithmen angewandt werden. Im Rahmen dieser Arbeit wurde die χ^2-Diskretisierung und die χ-Merge-Diskretisierung sowie Equal-Width-Binning und Equal-Frequency-Binning näher betrachtet und auf den Datensatz angewandt. Die χ-Merge-Diskretisierung betrachtet zunächst jedes Datenelement als ein eigenes Intervall und führt für jedes benachbarte Intervall den χ^2-Test durch. Die Intervalle mit den kleinsten χ^2-Werten werden anschließend miteinander verschmolzen. Dieser Prozess wird solange durchgeführt bis eine Abbruchbedingung erfüllt ist [HK2006].

Das Equal-Width-Binning unterteilt die Daten so, dass die Intervalle etwa gleichbreit sind. Das Equal-Frequency-Binning hingegen unterteilt die Daten so, dass jedes Intervall etwa die gleiche Anzahl an eindeutigen Werten umfasst.

Für den gegebenen Produktionsdatensatz wurde schließlich das Equal-Frequency-Binning gewählt, welches eine geringe Laufzeit vorweist und die Daten zuverlässig in eine bestimmte Anzahl an Untergruppen unterteilt. Hierfür wurde das Package *binr* ausgewählt. Die Diskretisierung erfolgt mit dem Befehl *bins* und gibt die Intervalle und Cutpoints zurück. Mit dem Befehl *cut* werden schließlich die Spalten des Datensatzes einzeln diskretisiert.

Die Anzahl der Untergruppen beträgt im Rahmen dieser Bearbeitung fünf, bis auf bei Buffer1Max und Buffer2Max, da die Werte dort bereits diskret vorlagen und daher mit den gegebenen Werten (zwei und sechs) weitergearbeitet wurde. Die Übersicht über alle Klassengrößen ist in *Anhang C: Produktionsdatensatz - Intervalle* zu finden.

Zufälliger Trainingsdatensatz

Nach der Vorverarbeitung der Daten werden die Daten nun in einen zufälligen Tranings-datensatz und einen Testdatensatz unterteilt.

```
randomSample = function(df, n){return (df[sample(nrow(df), n),])}
traindata <- randomSample(dataset, 2500)
testdata <- randomSample(dataset, 50000)
```

Mit der Verwendung des Trainingsdatensatzes kann der Rechenaufwand für die Erstellung der Modelle gering gehalten werden. Außerdem ist die Validierung der Modelle aussagekräftiger, da ein unabhängiger und vorher nicht bekannter Datensatz verwendet wird.

6 Anwendung

6.1 Berechnung der SVM und Parametrierung

Um den Rechenaufwand zu verkleinern, wird eine exponentiell ansteigende Zahlenfolge für die Parameter γ und C gewählt. Mit den Parametern, die zu einem guten Ergebnis führen, werden anschließend innerhalb eines kleineren Intervalls weitere Berechnungen durchgeführt, sodass sich dem optimalen Wert angenähert wird. So kann innerhalb einer akzeptablen Rechenzeit die beste SVM-Parametrisierung berechnet werden. Die gefundenen Parameter werden schließlich auf einen ausreichend großen Trainingsdatensatz angewendet. Dabei ist allerdings darauf zu achten, dass diese nicht zu groß werden, da die Laufzeit mit der Größe der Datensätze stark ansteigt. Dieses Vorgehen wird Rastersuche genannt und in dieser Arbeit mit der Kreuzvalidierung kombiniert.

Für die Optimierung der Parametrisierung der SVM wird eine v-fache Kreuzvalidierung durchgeführt. Dafür wird der Trainingsdatensatz in v Untergruppen eingeteilt. Anschließend wird die mit $v-1$ Untermengen trainierte SVM an dem übrig gebliebenen Datensatz getestet. Durch diesen Prozess kann eine Überanpassung der SVM an den Trainingsdatensatz vermieden werden, da die SVM nie mit dem kompletten Trainingsdatensatz berechnet wird.

Für die Rastersuche für die optimale Parametrisierung wird die Zahlenfolgen $cost = \{10^{-5}, 10^{-3}, 10^{-1}, 10, 10^{3}, 10^{5}, 10^{7}\}$ und $gamma = \{10^{-7}, 10^{-5}, 10^{-3}, 10^{-1}, 10, 10^{3}, 10^{5}\}$ verwendet. Die Rastersuche wird mit einer 10-fachen Kreuzvalidierung durchgeführt. Als Trainingsdatensatz wird hierfür ein zufällig ausgewählter Datensatz mit 2.500 Datenelementen gewählt. Anschließend wird eine SVM mit der optimalen Parametrisierung erstellt und anhand eines zufällig ausgewählten Testdatensatzes mit 50.000 Datenelementen validiert. Den Anteil an korrekt klassifizierten Datenelementen beschreibt die Accuracy.

Das Vorgehen der Berechnung der SVM ist für alle Leistungskennzahlen ähnlich. Zunächst wird die *tune()*-Funktion für den vorgegebenen *cost*- und *gamma*-Bereich durchgeführt, somit werden die optimalen Parameter berechnet und durch die *print()*-Funktion angezeigt. Anschließend werden diese Werte in die *svm()*-Funktion zur Berechnung des Modells eingesetzt. Darauf folgend wird das Modell validiert und als Matrix die Einteilung in die verschiedenen Klassen dargestellt und als *.rds*-Datei abgespeichert.

Die beispielhafte Berechnung einer SVC für die Leistungskennzahl Buffer1Max inklusive Tuning läuft folgendermaßen ab:

```
SVC_Buffer1Max_tune <- tune(svm, Buffer1Max ~ LoadingVolume + Buffer1 +
Buffer2 + NumberOfCarriers, CarrierSpeedLoaded + CarrierSpeedEmpty +
Product1 + Product2 + Product3 + Product4 + Product5 + Product6 + Prod-
uct7 + Product8 + Product9 + Product10 + Product11 + Product12 + Prod-
uct13 + Product14 + Product15 + Product16 + Product 17 + Product18 +
Product19 + Product20 + Product21 + Product22 + Product23 + Product24 +
Product25 + Product 26 + Product 27 + Machine1Speed + Machine2Speed +
Machine3Speed + Machine4Speed, data = trainset, type = "C-
Classification", kernel = "radial", ranges = list(gamma = gamma_list,
cost = cost_list), scale = T)
```

```
print(SVC_Buffer1Max_tune)
plot(SVC_Buffer1Max_tune)
SVC_Buffer1Max <- svm(Buffer1Max ~ LoadingVolume + Buffer1 + Buffer2 +
NumberOfCarriers, CarrierSpeedLoaded + CarrierSpeedEmpty + Product1 +
Product2 + Product3 + Product4 + Product5 + Product6 + Product7 + Prod-
uct8 + Product9 + Product10 + Product11 + Product12 + Product13 + Prod-
uct14 + Product15 + Product16 + Product 17 + Product18 + Product19 +
Product20 + Product21 + Product22 + Product23 + Product24 + Product25 +
Product 26 + Product 27 + Machine1Speed + Machine2Speed + Machine3Speed
+ Machine4Speed, data = trainset, type = "C-Classification", kernel =
"radial", gamma = 0.01, cost = 100), scale = T)
validate(SVC_Buffer1Max)
saveRDS(SVC_Buffer1Max, "./Models/SVC_Buffer1Max.rds")
```

Durch die Durchführung des Tuning-Algorithmus für alle Parameter wurden die in *Tabelle 3* dargestellte Parametrisierung für die Klassifikation gewählt.

Leistungskennzahl	Cost	Gamma	Kernel
Throughput	100	0,01	Radiale Basis
CycleTime	10.000	0,001	Radiale Basis
Buffer1Max	100	0,01	Radiale Basis
Buffer2Max	1.000	0,01	Radiale Basis
Buffer1Mean	1.000	0,001	Radiale Basis
Buffer2Mean	100.000	0,001	Radiale Basis
Machine1Utilization	2	0,1	Radiale Basis
Machine2Utilization	1.000	0,01	Radiale Basis
Machine3Utilization	1.000	0,01	Radiale Basis
Machine4Utilization	1.000	0,01	Radiale Basis

Tabelle 3: Parametrierung der SVC

Für die Regression wurde für alle SVRs die in *Tabelle 4* dargestellte Parametrisierung gewählt. Die Anwendung des Tuning-Algorithmus war auf Grund des zu hohen Rechenaufwandes nicht möglich. Durch Ausprobieren verschiedener Werte ist die folgende Parametrisierung gewählt worden.

Leistungskennzahl	Cost	Gamma	Kernel
Throughput	1	0.02702703	Radiale Basis
CycleTime	1	0.02702703	Radiale Basis
Buffer1Mean	1	0.02702703	Radiale Basis
Buffer2Mean	1	0.02702703	Radiale Basis
Machine1Utilization	1	0.02702703	Radiale Basis
Machine2Utilization	1	0.02702703	Radiale Basis
Machine3Utilization	1	0.02702703	Radiale Basis
Machine4Utilization	1	0.02702703	Radiale Basis

Tabelle 4: Parametrierung der SVR

Bei der Regression könnte tatsächlich eine bessere durchschnittliche normierte Abweichung erreicht werden, wenn ein größeres Kostenbudget gewählt wird. Allerdings steigt dadurch auch die maximale normierte Abweichung, so dass sich für ein geringes Kostenbudget entschieden wurde.
Für alle Modelle werden als unabhängige Variablen alle 37 Stellgrößen gewählt, sodass bei jeder SVM alle Eingangsvariablen berücksichtigt werden.

6.2 Evaluation

Klassifikation

Die Bewertung der SVC-Modelle wird anhand einer Kreuztabelle vorgenommen, so lässt sich schnell erkennen, wie groß die Zielgenauigkeit des Klassifikationsalgorithmus ist. Des Weiteren wird berechnet wie groß der Anteil der richtig klassifizierten Elemente an allen Elementen ist. Diese Kennzahl ergeben sich durch die Summe der Diagonale der Kreuztabelle dividiert durch die Gesamtzahl der Datenelemente. Neben dieser Kennzahl wird noch bestimmt wie viele Elemente nur um eine Klasse falsch klassifiziert wurden. Hierfür wird neben der Diagonale ebenfalls noch die Summe aus den beiden Nebendiagonalen gebildet.

		Vorhergesagt					
		1	2	3	4	5	6
Korrekt	1	42467	277	0	0	0	0
	2	305	44787	191	23	0	0
	3	5	238	48190	2688	44	8
	4	0	34	2961	43375	3074	150
	5	0	0	311	4975	89408	9075
	6	0	0	21	861	10825	95707

Tabelle 5: Kreuztabelle für Buffer2Max

Tabelle 5: Kreuztabelle für Buffer2Max stellt die vorhergesagten Werte den korrekten Werten gegenüber und visualisiert so wie exakt die SVC klassifiziert. Für Buffer2Max ist das Ergebnis mit einer Genauigkeit von 90% sehr gut. Weitere 9% wurden nur um eine Klasse falsch klassifiziert. So liegen 99% der Werte innerhalb der korrekten Klasse ± einer Klasse. Je weiter die Werte von der Diagonalen abweichen, desto schlechter wurde klassifiziert. In den Ecken sind hier keine Werte zu finden.

Leistungskennzahl	Genauigkeit	Genauigkeit ± 1 Klasse
Throughput	79,93%	99,32%
CycleTime	77,73%	99,25%
Buffer1Max	75,22%	99,13%
Buffer2Max	86,51%	99,45%
Buffer1Mean	82,65%	---
Buffer2Mean	90,98%	99,64%
Machine1Utilization	80,96%	99,09%
Machine2Utilization	86,38%	99,86%
Machine3Utilization	85,84%	99,75%
Machine4Utilization	87,83%	99,88%

Tabelle 6: Genauigkeit der SVC

Regression

Die Exaktheit der Regression wird mit der normierten mittleren absoluten Abweichung verglichen. Hierfür wird für jedes vorhergesagte Element die absolute Abweichung berechnet. Diese wird durch Division durch den korrekten Wert normiert, so dass die für alle Leistungskennzahlen vergleichbar sind. Neben dieser Kennzahl ist auch noch die Verteilung der mittleren absoluten Abweichung interessant. Diese wird mit einem Histogramm dargestellt. Als weiteres Kriterium wird die maximale normierte absolute Abweichung in die Bewertung der SVR einbezogen.

Formel 12: Normierte absolute Abweichung nach [HK2006]

$$\overline{nae} = \frac{1}{n}\sum_{1}^{n}\frac{|actual - predicted|}{actual}$$

$$\max(nae) = max\left(\frac{|actual_i - predicted_i|}{actual_i}\right)$$

Die Verteilung wird als Histogramm mit konstanten Klassengrößen 0,05 visualisiert.

Für die Regression wurden die Leistungskennzahlen Buffer1Max und Buffer2Max nicht berücksichtigt, da diese diskrete Werte darstellen und eine Klassifikation sinnvoller ist.

Für die Evaluation wurde als Testsatz der gesamte gegebene Produktionsdatensatz gewählt. Somit wurden die SVRs mit 400.000 Datenelementen evaluiert.
Mit den gewählten Parametern liegt die Abweichung der SVR bei der Leistungskennzahl Throughput für den gesamten Produktionsdatensatz im Mittel bei 4,1%. Allerdings weicht der Wert mit der größten Abweichung um 58,4% ab. In „Abbildung 5: Histogramm normierte absolute Abweichung Throughput“ lässt sich erkennen, dass ca. 75% der vorhergesagten Werte um weniger als 5% vom korrekten Wert abweichen. Allerdings weichen über 6% der von der SVR berechneten Werte um mehr als 10% von den eigentlichen Werten ab.

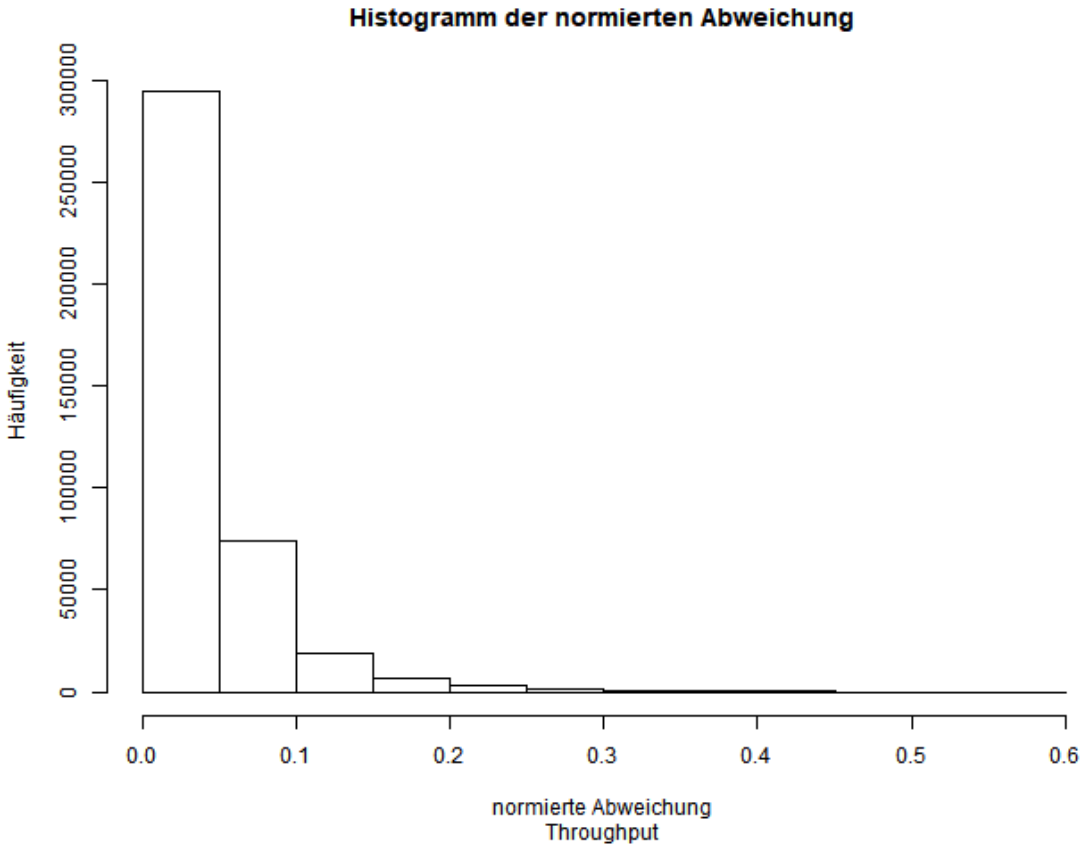


Abbildung 5: Histogramm normierte absolute Abweichung Throughput

Alle Histogramme der Verteilung der normierten absoluten Abweichung sind im Anhang zu finden.
In *Abbildung 6: Evaluation der SVR* werden die Abweichungen der acht SVR-Modelle verglichen. Es ist erkennbar, dass die Modelle für die Leistungskennzahlen Throughput, CycleTime und Machine1Utilization eine sehr geringe Abweichung aufweisen. Die Modelle für Buffer1Mean und Buffer2Mean hingegen sowohl eine hohe mittlere Abweichung von 0,2 bzw. 0,1 als auch eine hohe maximale Abweichung von 4 bzw. 3 auf. Für diese Leistungskennzahlen sollten die SVC genutzt werden, da hier hohe Genauigkeiten erzielt werden. Für die Leistungskennzahlen Machine2Utilization, Machine3Utilization und Machine4Utilization sind die mittleren Abweichungen von 0,05 bis 0,06 zwar auch zu beachten, allerdings befinden diese sich noch in einem akzeptablen Rahmen.

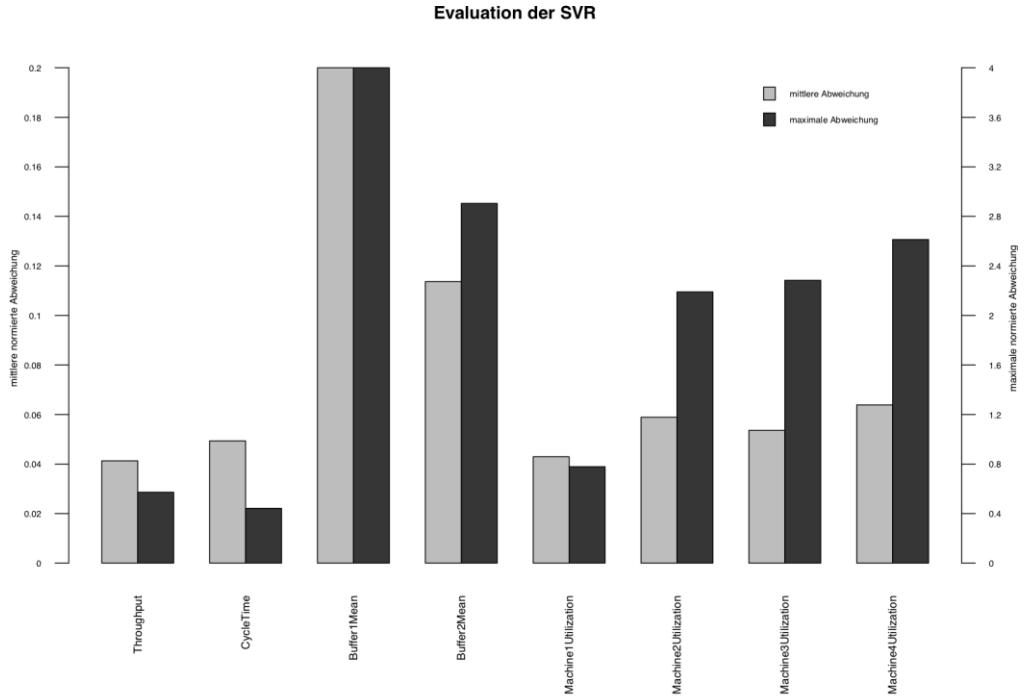


Abbildung 6: Evaluation der SVR

Leistungskennzahl	Mittlere Abweichung	Maximale Abweichung
Throughput	5,58%	121,32%
CycleTime	7,11%	83,78%
Buffer1Max	23,63%	596,64%
Buffer2Mean	16,54%	339,39%
Machine1Utilization	6,41%	117,80%
Machine2Utilization	9,00%	298,19%
Machine3Utilization	7,95%	313,17%
Machine4Utilization	9,66%	390,04%

Tabelle 7: mittlere und maximale prozentuale Abweichung der SVR

Vergleich Klassifikation und Regression

Im Anschluss an die Regression wurden die vorhergesagten Werte wiederum diskretisiert. Dafür wurden die gleichen Intervalle wie bei der Diskretisierung genutzt. Somit lassen sich die mit der Regression vorhergesagten Werte in den gleichen Klassen darstellen, wie die Trainingsdaten für die Klassifikation. So können dann die korrekten Klassen den von der Regression berechneten Klassen anschließend im Form einer Kreuztabelle gegenübergestellt werden. In Folge dessen ist ein Vergleich der beiden Algorithmen gut durchzuführen und somit kann die Regression neben der Abweichung noch auf einer weiteren Basis bewertet werden.

Exemplarisch wird diese Auswertung an der Leistungskennzahl Throughput aufgezeigt, da diese Variable die geringste mittlere Abweichung aufweist.

Throughput - Regression						
		Vorhergesagt				
		1	2	3	4	5
Korrekt	1	0	899	11.353	24.607	43.151
	2	777	15.533	20.500	22.890	20.316
	3	13.104	21.344	18.386	17.176	9.993
	4	24.352	21.836	15.784	14.624	3.393
	5	37.354	19981	15.029	7.024	594

Throughput - Klassifikation						
		Vorhergesagt				
		1	2	3	4	5
Korrekt	1	71.295	8.694	412	13	0
	2	8.140	60.689	11.419	512	34
	3	313	9.561	59.118	10.363	489
	4	24	171	12.613	58.642	7.637
	5	0	24	710	9.144	69.983

Tabelle 8: links: Kreuztabelle der Regression, rechts: Kreutabelle der Klassifikation

Wie in *Tabelle 8* zu sehen ist, sind die Ergebnisse bei dieser Leistungskennzahl deutlich schlechter als bei der Klassifikation, obwohl die mittlere sowie die maximale Abweichung gering sind. Die Regression weist eine Genauigkeit von 12,28% auf, die Klassifikation hingegen eine Genauigkeit von 79,93%. Dieses Ergebnis könnte teilweise durch die Klasseneinteilung bedingt sein, allerdings wird trotzdem die Verwendung des Klassifikationsalgorithmus empfohlen.

7 Kritische Würdigung

Wie sich im Lauf dieser Arbeit herausgestellt hat ist die richtige Vorverarbeitung der Daten ein essenzieller Schritt auf dem Weg zu einer erfolgreichen Modellbildung. Des Weiteren wurde aufgezeigt, dass der Algorithmus Support Vector Machine sich gut auf Datensätze anwenden lässt, deren Leistungskennzahlen aus diskreten Daten bestehen. In diesem Fall kann die Klassifikation genutzt werden. Im gegebenen Datensatz ist dies bei den Leistungskennzahlen Buffer1Max und Buffer2Max der Fall. Bei kontinuierlichen Daten lässt sich die Klassifikation nur nach Durchführung einer Vorverarbeitung durchführen. Diese umfasst die Diskretisierung der Daten und ggf. Umwandlung in Faktoren. Ein weiterer wichtiger Teil der Vorverarbeitung ist die Normierung der Stellgrößen, damit der Einfluss dieser Variablen die Modellbildung nicht zu stark übersteigt. Dennoch ist über die Parameter Cost und Gamma eine Einflussnahme auf die Anpassung des Modells an den Datensatz möglich. So kann sowohl eine Unteranpassung als auch eine Überanpassung vermieden werden. Ein Vorteil, der auch Nachteile birgt, ist, dass durch die verschiedenen Arten von SVMs mit unterschiedlichen Parametern sehr viele unterschiedliche Varianten existieren. Hieraus ist eine passende Kombination aus SVM-Typ und Parametrisierung auszuwählen. Für eine optimale Parametrisierung wäre es hier vorteilhaft, wenn die tune-Funktion neben dem Kostenbudget und weiteren Parametern ebenfalls die verschiedenen Kernel berücksichtigt.

Für nicht diskrete oder diskretisierte Daten ist ebenfalls die Anwendung von SVMs möglich. Die Regression hat einen höheren Aufwand und teilweise sehr große Abweichungen, daher sollten nicht nur einzelne Daten für die Regression verwendet werden, sondern ein Datensatz, so dass diese großen Abweichungen durch andere Berechnungen ausgeglichen werden können.

Literaturverzeichnis

[ABR1964] Aizerman, M. A.; Braverman, È. M.; Rozonèr, L. I.: *Theoretical foundation of potential functions method in pattern recognition*. In: Avtomat. I Telemekh. Vol. 25, Nr. 6, 1964, S. 917-936

[Al2003] Al-Laham, A.: *Organisationales Wissensmanagement – eine strategische Perspektive*.1.Aufl., Vahlen Verlag, München, 2003

[Al2010] Alpaydin, C.: *Introduction to Machine Learning*. 2. Aufl., MIT Press, Cambridge, MA, 2010

[Bi2008] Bishop, C.: *Pattern Recognition and Machine Learning.* Information Science and Statistics, Springer, Berlin, 2008

[BV2008] Bankhofer, U.; Vogel, J.: *Datenanalyse und Statistik.* 1. Aufl., Gabler Verlag, Wiesbaden, 2008

[CC2011] Chang, C.-C.; Lin, C.-J.: *LIBSVM:a library for support vector machines.* In: ACM Transactions on Intelligent Systems and Technology. Vol. 2, Nr. 3, 2011, S. 27:1-27:27

[FPS1996] Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P.: *From Data Mining to Knowledge Discovery in*. In: AI Magazine. Nr. 17, 1996, S. 37-54.

[Fi1936] Fisher, R. A.: *The use of multiple measurements in taxonomic problems*. In: Annals of Eugenics. Nr. 7, 1936, S. 179-188.

[Ha2009] Hamel, L.; *Knowledge Discovery with Support Vector Machines*. John Wiley & Sons, New Jersey, 2009

[HK2006] Han, J.; Kamber, M.: *Data Mining Concepts and Techniques*. 2. Aufl., Elsevier, Morgan Kaufmann, Amsterdam, Boston, San Francisco, CA, 2006

[Ja+2013] James, G.; Witten, D.; Hastie, T.; Tibshirani, R.: *An Introduction to Statistical Learning with Applications in R*, 1. Aufl., Springer, New York, Heidelberg, Dordrecht, London, 2013

[Kü1999] Küppers, B.: *Data Mining in der Praxix, ein Ansatz zur Nutzung der Potenziale von Data Mining im betrieblichen Umfeld.* 1. Aufl., Lang, Frankfurt am Main, 1999

[Me2017] Meyer, D.: *Support Vector Machines * The Interface to libsvm in package e1071*. FH Technikum, Wien, 2017

[Ro1958] Rosenblatt, F.; T*he perceptron: A probalistic model for information storage and organization in the brain.* In: Psychological Review. Vol. 65, Nr. 6, 1958 S. 386-408.

[VC1995] Vapnik, V. N.; Cortes, C: *Support-Vector Networks*. In: Machine Learning. Nr. 20, 1995, S. 273-297

[VL1963] Vapnik, V. N.; Lerner, A. Y.: *Pattern Recognition using generalized portraits.* In: Automation and Remote Control. Vol. 24, Nr. 6, 1963, S. 709-715

[Va995] Vapnik, V. N.: *The Nature of Statistical Learning Theory*. 2. Aufl., Springer, Berlin, 1995

Anhang A: Herleitung linearer SVM

$\langle \vec{w}, \vec{x} \rangle$ beschreibt das Skalarprodukt aus den Vektoren w und x.
Ein n-dimensionaler Unterraum wird durch folgende Gleichung beschrieben, beispielsweise eine Geradengleichung im zweidimensionalen Raum.

(1) $f(x) = \langle \vec{w}, \vec{x} \rangle + b = \sum_{i=1}^{N} w_i x_i + b$

Mit der Bedingung, dass die dem Unterraum am nächsten liegenden Punkte gleich weit entfernt sind, ergeben sich folgende Gleichungen:

(2) $\langle \vec{w}, \overrightarrow{x_1} \rangle + b = +1$

(3) $\langle \vec{w}, \overrightarrow{x_2} \rangle + b = +1$

Durch Umformung der Gleichungen ergibt sich der Abstand zwischen den Stützvektoren 1 und 2:

(4) $\langle \vec{w}, (\overrightarrow{x_1} - \overrightarrow{x_2} \rangle = 2$

Durch Normieren des Vektors $\vec{w}$ ergibt sich der Abstand zwischen den zwei Stützvektoren in Richtung des zur Hyperebene orthogonalen Vektors $\vec{w}$ zu $\frac{2}{\|w\|}$.

(5) $\langle \frac{\vec{w}}{\|w\|}, (\overrightarrow{x_1} - \overrightarrow{x_2} \rangle = \frac{2}{\|w\|}$

Der Rand (linke Seite von (5)) wird also maximiert, indem $\|w\|$ minimiert wird.
Somit muss für die Berechnung der optimalen Hyperebene das Optimierungsproblem gelöst werden, den in (5) beschriebenen Abstand zwischen den Stützvektoren zu maximieren.
Als Entscheidungsfunktion wird die signum-Funktion herangezogen, welche folgendermaßen aussieht:

(6) $h(x) = sgn\big(f(x)\big) = \begin{cases} +1 \; falls \; f(x) > 0 \\ -1 \; sonst \end{cases}$

Im Fall der Stützvektoren ergibt sich damit:

(7) $f(x_i) = sgn(\langle \vec{w}, \overrightarrow{x_i} \rangle + b) = y_i$

Wenn die Datenelemente also korrekt klassifiziert wurden, führt das dazu, dass folgende Funktion immer positiv ist:

(8) $y_i(\langle \vec{w}, \overrightarrow{x_i} \rangle + b) \geq 1 \quad \forall \, i = 1, \dots, N$

Das bereits beschriebene Optimierungsproblem wird mit Lagrange gelöst:

(9) $L(w, b, \propto) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} \propto_i \, (y_i(\langle \vec{w}, \overrightarrow{x_i} \rangle + b) - 1)$

mit $\propto_i = \propto_1 \dots \propto_N$ und $\propto_i \geq 0$

Nun wird bzgl. $\propto$ maximiert und bzgl. w und b minimiert, somit:

(10) $\frac{\delta}{\delta b} L(w, b, \propto) = 0$

(11) $\sum_{i=1}^{N} \propto_i \, y_i = 0$

und:

(12) $\frac{\delta}{\delta w} L(w, b, \propto) = 0$

(13) $w = \sum_{i=1}^{N} \propto_i y_i x_i$

Laut den Kuhn-Tucker-Bedingungen gilt für den Sattelpunkt:

(14) $\propto_i [y_i(\langle \vec{w}, \vec{x_i} \rangle + b) - 1] = 0 \quad \forall\, i = 1, \dots, N$

Somit gilt im Optimum:

(15) $\propto_i = 0 \quad$ oder $\quad y_i(\langle \vec{w}, \vec{x_i} \rangle + b) = 1$

Mit (13) lässt sich erkennen, dass nur $\propto_i > 0$ Einfluss auf die optimale Lösung haben. Diese Elemente liegen auf dem Rand und sind Stützvektoren.
Es ergibt sich:

(16) $\underset{\propto \in \mathbb{R}^n}{maximiere} \quad \sum_{i=1}^{N} \propto_i - \frac{1}{2} \sum_{i,j=1}^{N} \propto_i \propto_j \; y_i y_j \, \langle x_i, x_j \rangle$

unter den Bedingungen:

(17) $\propto_i \geq 0 \quad \forall i = 1, \dots, N$ und $\sum_{i=1}^{N} \propto_i y_i = 0$

Somit wurden die Lagrange-Multiplikatoren $\propto_i$ und damit die Stützvektoren berechnet.
Mit (7) und (13) führt dies zu der Entscheidungsfunktion:

(18) $f(x) = sgn(\sum_{i=1}^{N} \propto_i y_i \langle x, x_i \rangle + b)$

Wie sich erkennen lässt, ist die Entscheidungsfunktion nur von den Skalarprodukten der unabhängigen Merkmale der Stützvektoren abhängig, da nur Datenelemente mit $\propto_i > 0$ Einfluss haben.

Anhang B: Herleitung Soft Margin Classifier

Durch Abschwächen der Randbedingungen kann der Rechenaufwand von SVMs gesenkt werden. Dies geschieht durch das Zulassen einer gewissen Zahl von Ausreißern. Diese werden durch die Einführung einer Schlupfvariablen $\xi_i \geq 0$, eines Fehlergewichtes γ und eines Fehlerbudgets C kontrolliert.
Es ergibt sich:

$$y_i(\langle \vec{w}, \vec{x_i} \rangle + b) \geq 1 - \xi_i \quad \forall\, i = 1, \dots, N$$

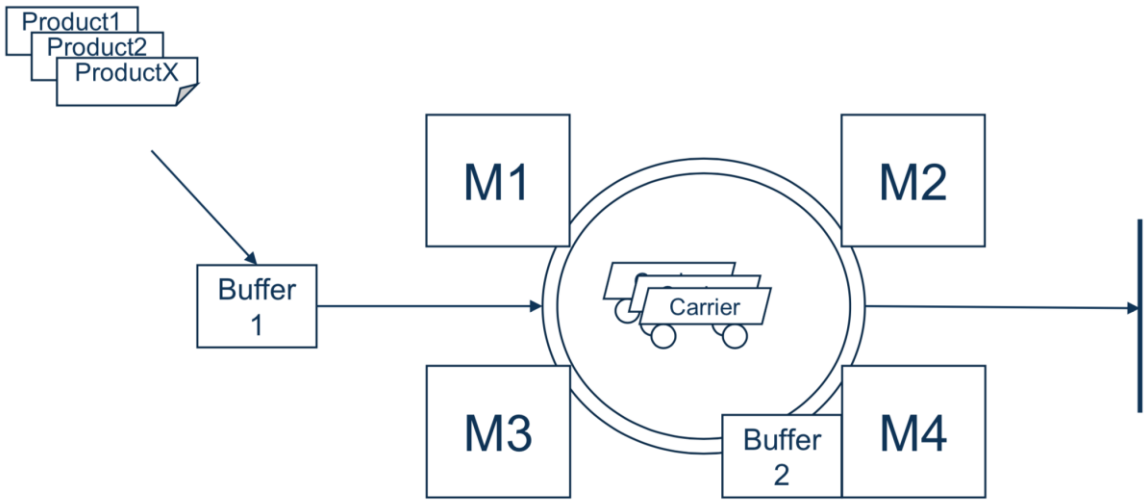
Bei korrekter Klassifikation ergibt sich $\xi_i = 0$, wenn das Datenelement innerhalb des Randes ist $0 < \xi_i < 1$, und falls das Datenelement auf der falschen Seite ist $\xi_i > 1$.
Wenn für SVMs statt der Hard Margin die Soft Margin gewählt wird, so kann eine Überanpassung an den Trainingsdatensatz vermieden werden, da nicht jeder Datenpunkt, welcher einen neuen Stützvektor bilden würde, Einfluss auf die Hyperebene hat.

Anhang C: Produktionsdatensatz - Intervalle

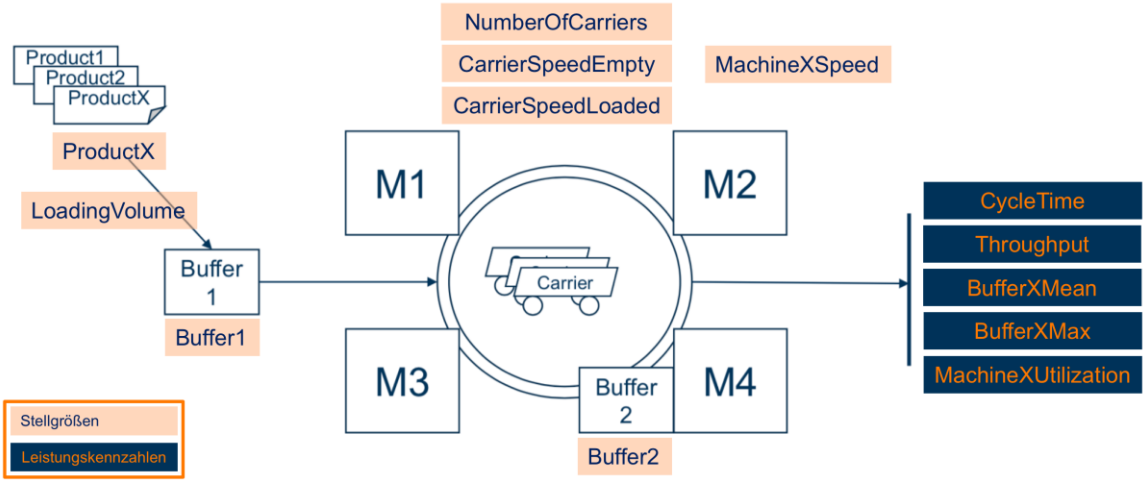
	Attributname	Datentyp	Intervall
Stellgrößen	LoadingVolume	Ganzzahlig	{100; 200}
	Buffer1, Buffer2	Ganzzahlig	{1; 10}
	NumberOfCarriers	Ganzzahlig	{3; 8}
	CarrierSpeedLoaded	Gleitkommazahl	{0,07; 0,10}
	CarrierSpeedEmpty	Gleitkommazahl	{0,77; 1,15}
	ProductX [1-27]		{0,0011; 0,82}
	MachineXSpeed [1-4]	Gleitkommazahl	{0,7; 2,0}
Leistungskennzahlen	Throughput	Ganzzahlig	{165; 803}
	Buffer1Mean, Buffer2Mean	Gleitkommazahl	{0,01; 1,43} {0,38; 4,84}
	Buffer1Max, Buffer2Max	Ganzzahlig	{1; 2} {1; 6}
	CycleTime	Gleitkommazahl	{42,55; 347,06}
	Machine1Utilization Machine2Utilization Machine3Utilization Machine4Utilization	Gleitkommazahl	{13,32; 99,99} {2,97; 99,99} {2,98; 99,79} {2,51; 98,95}

Anhang D: Systemübersicht

Systemübersicht



Systemübersicht mit Stellgrößen und Leistungskennzahlen



Anhang E: Klassen der Leistungskennzahlen

Klassen(größen) und Intervalle

Leistungskennzahl	Klasse 1	Klasse 2	Klasse 3	Klasse 4	Klasse 5	Klasse 6
Throughput	(0; 296,5]	(296,5; 348,5]	(348,5; 400,5]	(400,5; 461,5]	(461,5; ∞)	---
CycleTime	(0; 88,5]	(88,5; 103,4]	(103,3; 117,6]	(117,6; 137,1]	(137,1; ∞]	---
Buffer1Mean	(0; 0,05]	(0,05; 0,08]	(0,08; 0,1]	(0,1; 0,13]	(0,13; ∞]	---
Buffer2Mean	(0; 1,78]	(1,78; 2,32]	(2,32; 2,83]	(2,83; 3,45]	(3,45; ∞]	---
Buffer1Max	1	2	---	---	---	---
Buffer2Max	1	2	3	4	5	6
Machine1Utilization	(0; 54,32]	(54,32; 70,22]	(70,22; 84,64]	(84,64; 96]	(96; ∞]	---
Machine2Utilization	(0; 28,15]	(28,15; 28,83]	(28,83; 50,66]	(50,66; 67,41]	(67,41; ∞]	---
Machine3Utilization	(0; 43,66]	(43,66; 57,59]	(57,59; 71,65]	(71,65; 88,3]	(88,2; ∞]	---
Machine4Utilization	(0; 29]	(29; 40,96]	(40,96; 52,66]	(52,66; 69,22]	(69,22; 52,7]	---

Anhang F: Evaluation von SVC und SVR

Zusammenfassung: Regression

Leistungskennzahl	Cost	Gamma	Mittlere Abweichung	Maximale Abweichung
Throughput	1	0,02702703	5,58%	121,32%
CycleTime	1	0,02702703	7,11%	83,78%
Buffer1Mean	1	0,02702703	23,63%	596,64%
Buffer2Mean	1	0,02702703	16,54%	339,39%
Machine1Utilization	1	0,02702703	6,41%	117,80%
Machine2Utilization	1	0,02702703	9,00%	298,19%
Machine3Utilization	1	0,02702703	7,95%	313,17%
Machine4Utilization	1	0,02702703	9,66%	390,04%

Zusammenfassung: Klassifikation

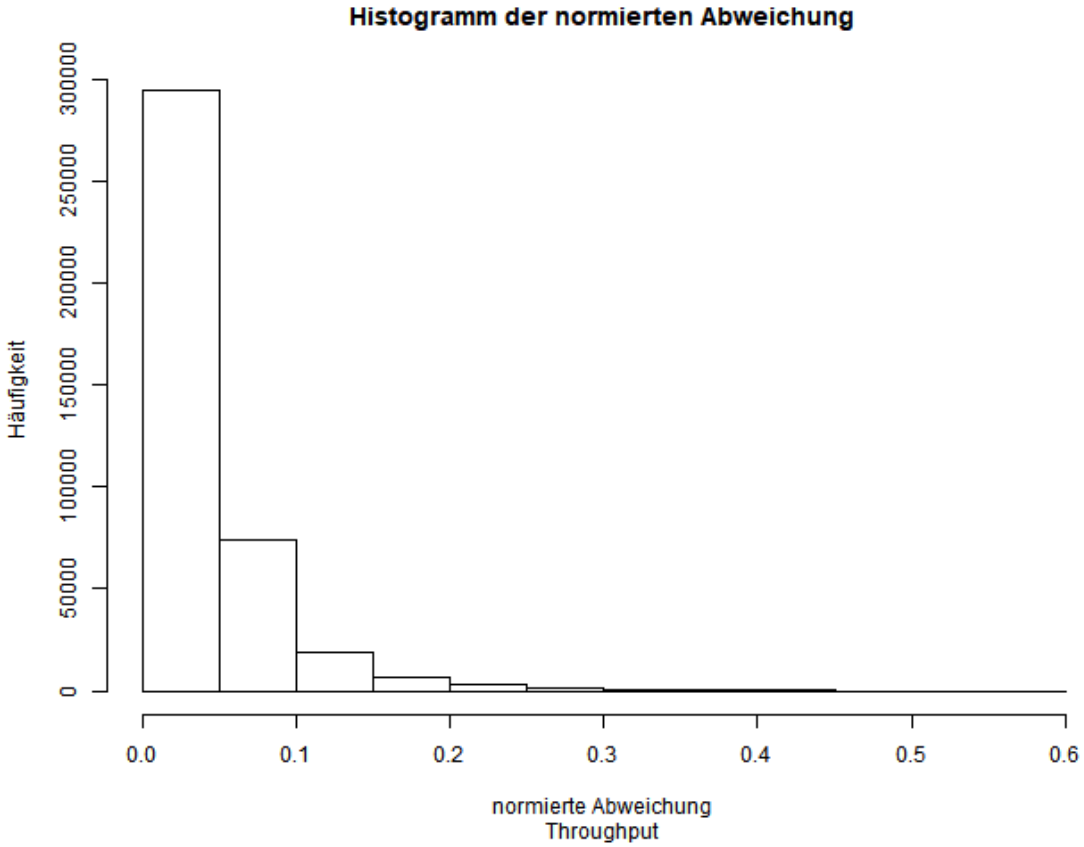
Leistungskennzahl	Cost	Gamma	Accuracy	Accuracy±1
Throughput	100	0,01	79,93%	99,32%
CycleTime	10.000	0,001	77,73%	99,25%
Buffer1Mean	1.000	0,001	75,22%	99,13%
Buffer2Mean	100.000	0,001	86,51%	99,45%
Buffer1Max	100	0,01	82,65%	---
Buffer2Max	1.000	0,01	90,98%	99,64%
Machine1Utilization	2	0,1	80,96%	99,09%
Machine2Utilization	1.000	0,01	86,38%	99,86%
Machine3Utilization	1.000	0,01	85,84%	99,75%
Machine4Utilization	1.000	0,01	87,83%	99,88%

Throughput

Klassifikation

Throughput						
		Predicted				
		1	2	3	4	5
Actual	1	71.295	8.694	412	13	0
	2	8.140	60.689	11.419	512	34
	3	313	9.561	59.118	10.363	489
	4	24	171	12.613	58.642	7.637
	5	0	24	710	9.144	69.983

Regression



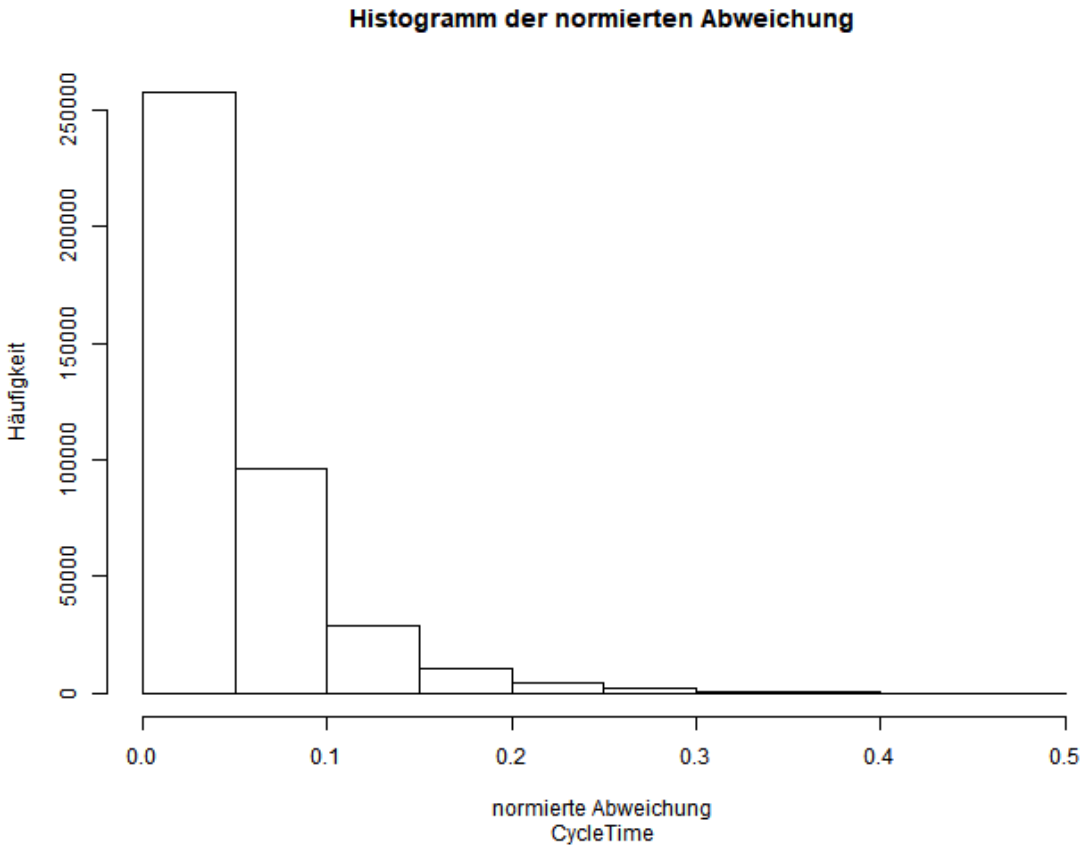
Mittlere normierte absolute Abweichung	0,0410236517821799
Maximale normierte absolute Abweichung	0,584798131061289

CycleTime

Klassifikation

CycleTime						
		Predicted				
		1	2	3	4	5
Actual	1	71.569	8.236	188	17	0
	2	8.143	59.165	11.883	781	44
	3	303	11.417	55.257	12.597	429
	4	34	547	12.743	56.999	9.666
	5	0	73	571	11.425	67.913

Regression



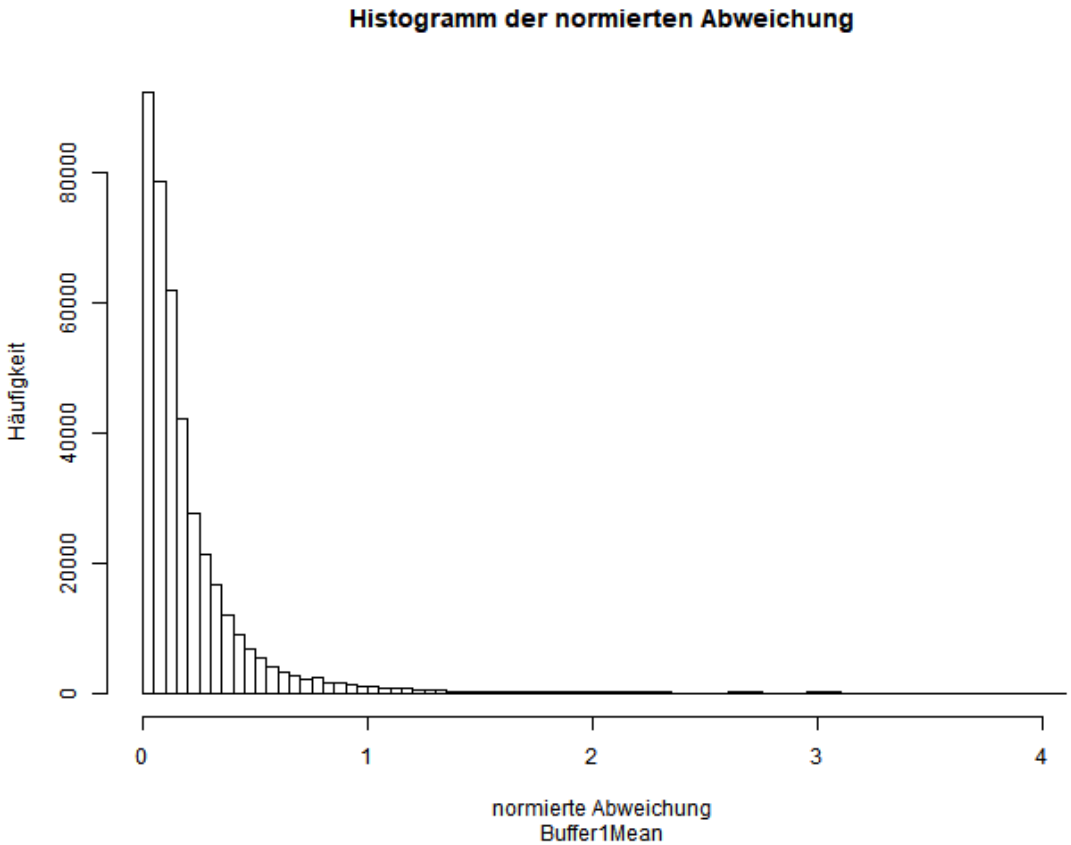
Mittlere normierte absolute Abweichung	0,0711060695256906
Maximale normierte absolute Abweichung	0,837839987019846

Buffer1Mean

Klassifikation

Buffer1Mean						
		Predicted				
		1	2	3	4	5
Actual	1	79.242	11.868	121	23	0
	2	16.212	39.907	9.540	1.699	0
	3	75	2.789	44.509	22.789	86
	4	24	1.275	8.967	86.813	1.831
	5	4	72	106	21.637	50.411

Regression



Mittlere normierte absolute Abweichung	0,236270680503751
Maximale normierte absolute Abweichung	5,96638504269851

Buffer1Max

Klassifikation

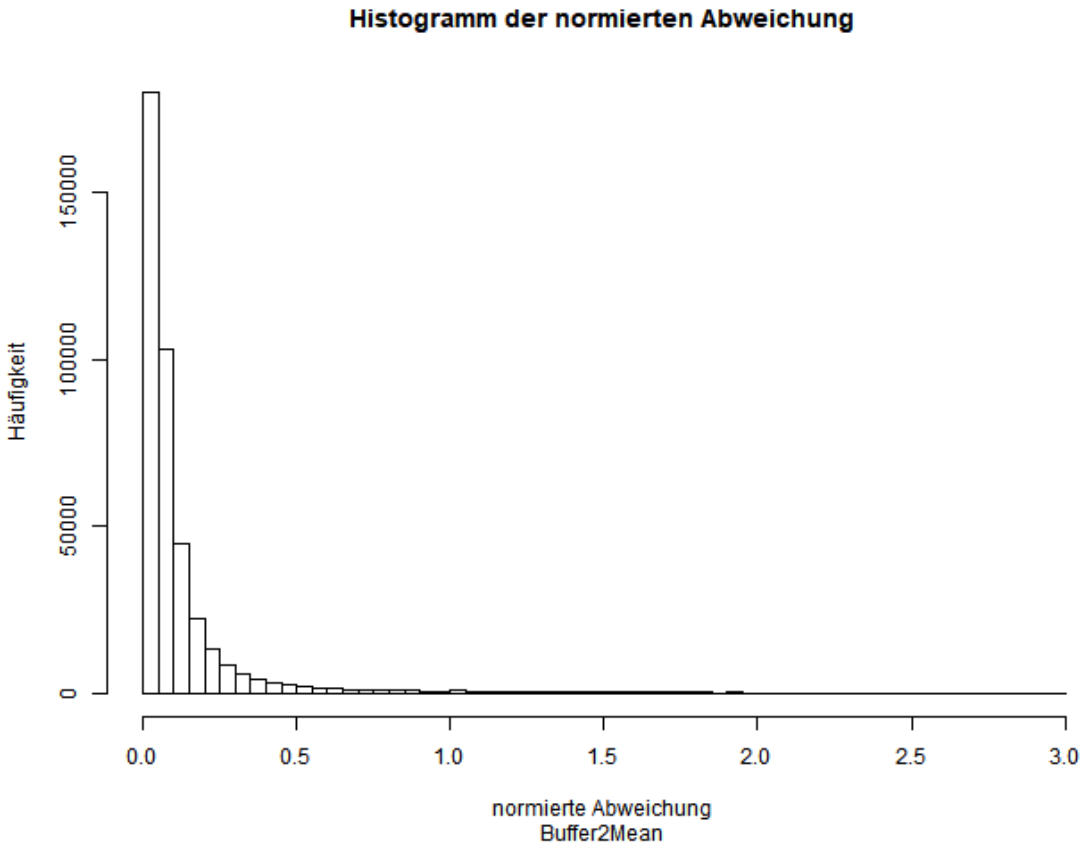
Buffer1Max			
		Predicted	
		1	2
Actual	1	190.530	30.016
	2	39.279	140.075

Buffer2Mean

Klassifikation

Buffer2Mean						
		Predicted				
		1	2	3	4	5
Actual	1	75.820	4.258	41	27	0
	2	4.490	70.085	5.830	692	29
	3	317	5.858	64.215	8.521	274
	4	68	452	8.165	64.353	6.642
	5	4	69	228	7.977	71.585

Regression



Mittlere normierte absolute Abweichung	0,165449208917654
Maximale normierte absolute Abweichung	3,39387705778112

Buffer2Max

Klassifikation

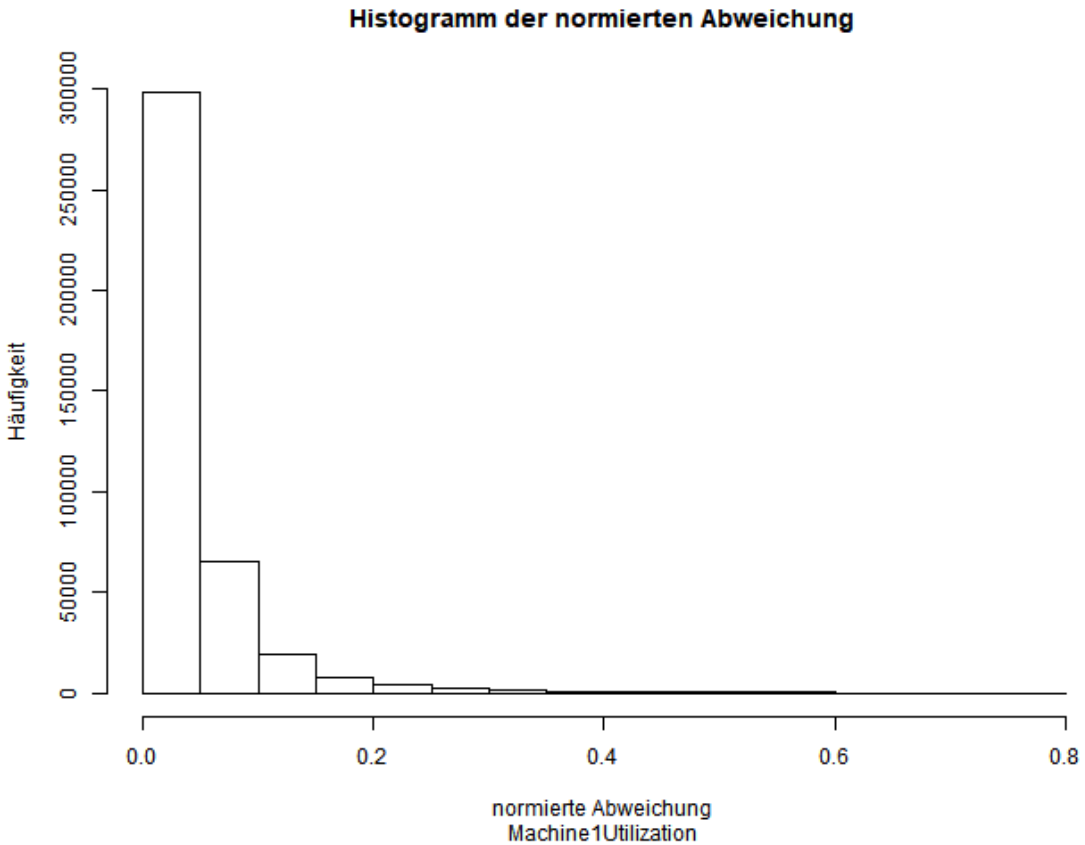
Buffer2Max							
		Predicted					
		1	2	3	4	5	6
Actual	1	42.467	277	0	0	0	0
	2	305	44.787	191	23	0	0
	3	5	238	48.190	2.688	44	8
	4	0	34	2.961	43.375	3.074	150
	5	0	0	311	4.975	89.408	9.075
	6	0	0	21	861	10.825	95.707

Machine1Utilization

Klassifikation

Machine1Utilization						
		Predicted				
		1	2	3	4	5
Actual	1	70.002	9.478	496	39	9
	2	7.325	61.766	10.404	440	79
	3	307	9.143	60.773	9.239	549
	4	29	936	10.281	61.229	7.503
	5	0	77	662	9.180	70.054

Regression



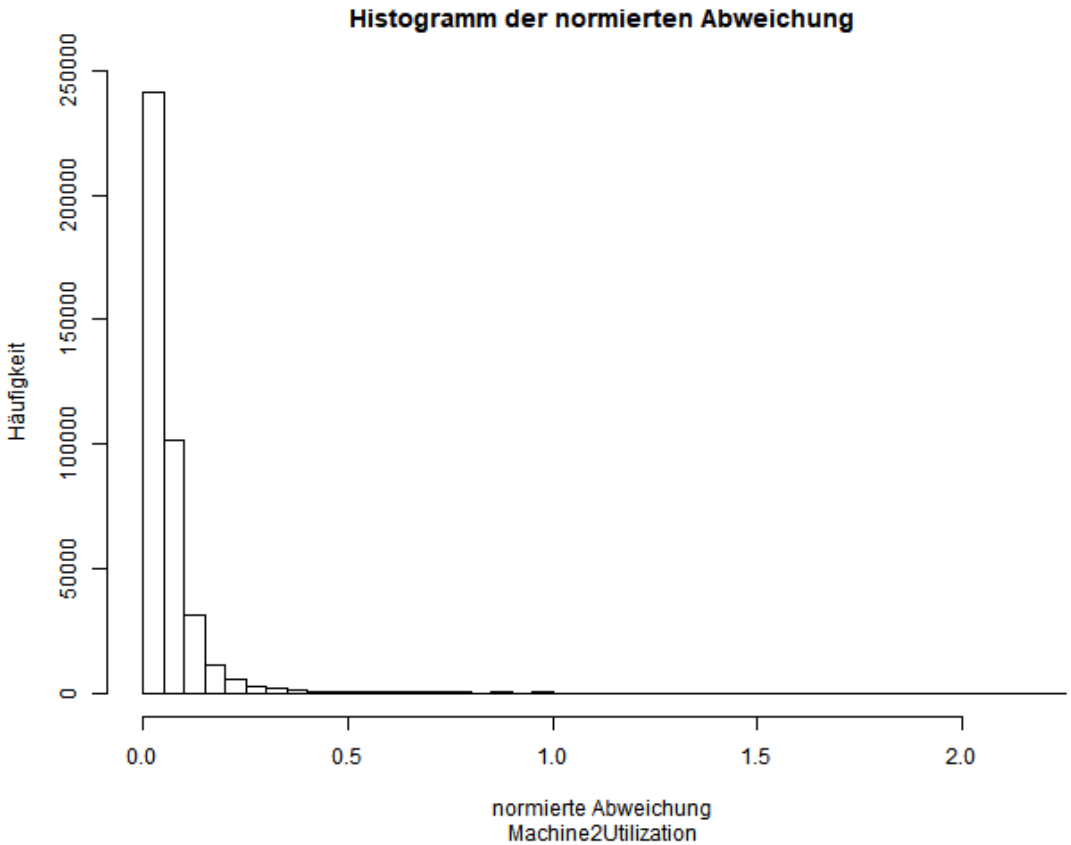
Mittlere normierte absolute Abweichung	0,064116554211621
Maximale normierte absolute Abweichung	1,17801229264222

Machine2Utilization

Klassifikation

Machine2Utilization						
		Predicted				
		1	2	3	4	5
Actual	1	74.401	5.568	44	0	0
	2	6.711	65.106	7.885	284	4
	3	51	6.542	66.065	7.219	160
	4	0	15	7.705	66.201	6.075
	5	0	0	0	6.217	73.747

Regression



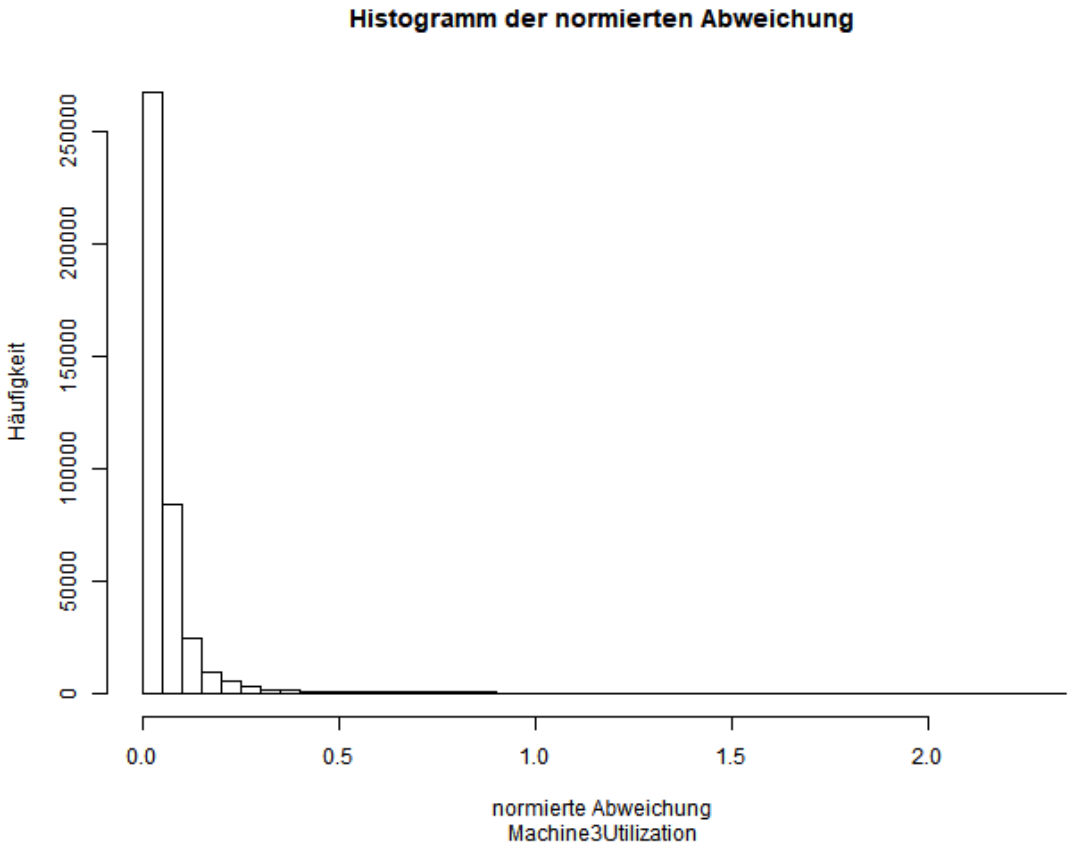
Mittlere normierte absolute Abweichung	0,0900126039177117
Maximale normierte absolute Abweichung	2,9818935573859

Machine3Utilization

Klassifikation

Machine3Utilization						
		Predicted				
		1	2	3	4	5
Actual	1	74.493	5.379	150	0	0
	2	5.430	65.299	9.170	124	16
	3	30	7.202	65.549	6.921	256
	4	0	37	8.677	65.111	6.180
	5	0	0	397	6.677	72.902

Regression



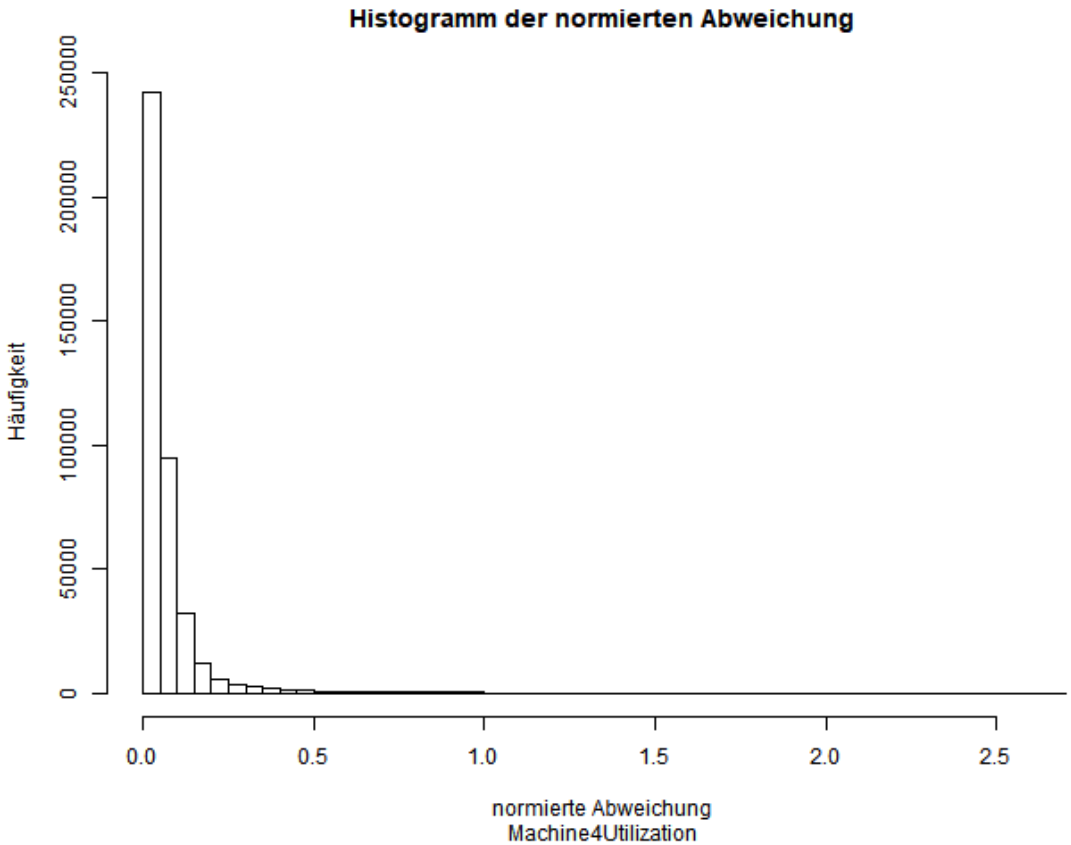
Mittlere normierte absolute Abweichung	0,0795108689618553
Maximale normierte absolute Abweichung	3,13172521509031

Machine4Utilization

Klassifikation

Machine4Utilization						
		Predicted				
		1	2	3	4	5
Actual	1	75.037	4.843	162	0	0
	2	3.750	68.512	7.743	42	0
	3	11	6.259	67.270	6.324	115
	4	0	50	8.264	66.011	5.618
	5	0	0	99	5.412	74.478

Regression



Mittlere normierte absolute Abweichung	0,0966409198442389
Maximale normierte absolute Abweichung	3,90043762816189

Anhang G: Liesmich-Datei

```
Mit der Date "Application.R" können die SVR- und SVC-Modelle auf
einen Datensatz angewendet werden, der als csv-Datei vorliegt.
Die Stellgrößen werden für die SVR vorbearbeitet und anschließend
als rds- und csv-Datei ausgegeben.
Im Folgenden wird die Anwendung beschrieben. Die Zeilen des
Skripts können chronologisch ausgeführt werden.
-------------------------------------------------------
1.) Öffnen Sie das R Script "Application.R" im Ordner Code.
1a.) Ggf. muss die Working Directory gesetzt werden. Hier wird
der Oberordner, in der Ordner mit den Daten liegt, angegeben.
(Hier: "C:/Users/User/Hauptseminar SVM/")
2.) Nun muss die Library e1071 geladen werden.
3.) Des Weiteren müssen die Funktionen preprocessing(),
svm.regression() und svm.classification() geladen werden.
4.) Nun wird das Verzeichnis eingegeben, in dem die csv-Datei mit
den Stellgrößen liegt.
5.) Des Weiteren werden die Verzeichnisse eingegeben, in denen
die Modelle abliegen. (Default: "./Model/Regression/" und
"./Model/Classification/".)
6.) Danach werden die vorherzusagenden Leistungskennzahlen als
Vektor mit character-Werten eingegeben.
6.) Anschließend werden die Stellgrßflen vorbearbeitet und nor-
miert.
7.) Nun können die Funktionen zur Regression und Klassifikation
ausgeführt werden.
8) Die Vorhersagen werden im gleichen Ordner wie die Inputdaten
als csv-Datei gespeichert.
-------------------------------------------------------
Verzeichnisstruktur:

Hauptseminar SVM
- Code
--- Application.R
- Data
--- csv-Datei
--- rds-Dateien
- Graphics
--- Histogram
- Model
--- Classification
--- Regression
--- Evaluation
-------------------------------------------------------
Die Modelle können mit Skript "Classification.R" und "Regressi-
on.R" berechnet werden.
Für die Erstellung der Modelle müssen die Daten ggf. mit "Prepro-
cessing.R" vorbearbeitet werden.
Anschlieflend können die Modelle mit "Evaluation.R" ausgewertet
werden.
1.) Erstellen der Verzeichnisstruktur
2.) Preprocessing.R
3.) Classification.R
4.) Regression.R
```

5.) Application.R
6.) Evaluation.R
Weitere Informationen und eine Einführung in die SVM-Methode sind im Hauptseminar "Machine Learning - Support Vector Machines" gefunden werden.

Kontakt: Tobias Rummelsberger
E-Mail: tobias.rummelsberger@tu-ilmenau.de
Mobil: +49 157 89256742

Anhang H: Code

Example

```
# WICHTIG: Library e1071 laden!
library(e1071)

############################################################
# Laden des Beispieldatensatzes
example.df <- read.csv2(paste("./Data/", "Beispieldaten.csv", sep =
""))
example.df$Y <- factor(example.df$Y)
saveRDS(example.df, "./Data/Beispieldatensatz.rds")

example.df <- readRDS("./Data/Beispieldatensatz.rds")

# Visualisieren des Beispieldatensatzes
plot(x = example.df$X1, y = example.df$X2, type = "p", pch = 4, xlab =
"X1", ylab = "X2", col = (as.numeric(example.df$Y)))

# Erstellen einer linearen Hard Margin SVM
example.svm <- svm(Y ~ X1+X2, data = example.df, kernel = "linear",
cost = 10000, gamma = 0.5)
plot(example.svm, data = example.df, formula = X2~X1, grid = 250,
svSymbol = 2, dataSymbol = 4, fill = T)

# Erstellen einer linearen Soft Margin SVM
example.svm <- svm(Y ~ X1+X2, data = example.df, kernel = "radial",
cost = 0.3, gamma = 0.01)
plot(example.svm, data = example.df, formula = X2~X1, grid = 250,
svSymbol = 2, dataSymbol = 4, fill = T, sub = "SVM linear Soft Margin")

# Tuning der SVM
example.svm.tune <- tune(svm, Y~X1+X2, data = example.df, kernel = "ra-
dial", ranges = list(cost = c(1:1000), gamma = c(0,1,0.1)))
print(example.svm.tune)
plot(example.svm.tune)
```

EDA – Exploratory Data Analysis

```
# Exploratory Data Analysis
# Zusammenfassung der Daten und Visualisierung mittels Boxplot-
Diagrammen
# Erfassen der Intervalle und Datentypen der Attribute
# Grundlage für die Vorverarbeitung der Daten (insb. Normalisierung,
Diskretisierung)

# Lesen des Datensatzes
Produktionsdaten <- readRDS("./Data/Produktionsdaten.rds")

# alle Daten
summary(Produktionsdaten)
str(Produktionsdaten)

# Stellgrößen
# LoadingVolume
boxplot(Produktionsdaten$LoadingVolume)
range(Produktionsdaten$LoadingVolume)
summary(Produktionsdaten$LoadingVolume)

# Buffer1
boxplot(Produktionsdaten$Buffer1)
range(Produktionsdaten$Buffer1)
summary(Produktionsdaten$Buffer1)

# Buffer2
boxplot(Produktionsdaten$Buffer2)
range(Produktionsdaten$Buffer2)
summary(Produktionsdaten$Buffer2)

# NumberOfCarriers
boxplot(Produktionsdaten$NumberOfCarriers)
range(Produktionsdaten$NumberOfCarriers)
summary(Produktionsdaten$NumberOfCarriers)

# CarrierSpeedLoaded
boxplot(Produktionsdaten$CarrierSpeedLoaded)
range(Produktionsdaten$CarrierSpeedLoaded)
summary(Produktionsdaten$CarrierSpeedLoaded)

# CarrierSpeedEmpty
boxplot(Produktionsdaten$CarrierSpeedEmpty)
range(Produktionsdaten$CarrierSpeedEmpty)
summary(Produktionsdaten$CarrierSpeedEmpty)
```

```
# Produktmix

# MachineXSpeed
# Machine1
boxplot(Produktionsdaten$Machine1Speed)
range(Produktionsdaten$Machine1Speed)
summary(Produktionsdaten$Machine1Speed)

# Machine2
boxplot(Produktionsdaten$Machine2Speed)
range(Produktionsdaten$Machine2Speed)
summary(Produktionsdaten$Machine2Speed)

# Machine3
boxplot(Produktionsdaten$Machine3Speed)
range(Produktionsdaten$Machine3Speed)
summary(Produktionsdaten$Machine3Speed)

# Machine4
boxplot(Produktionsdaten$Machine4Speed)
range(Produktionsdaten$Machine4Speed)
summary(Produktionsdaten$Machine4Speed)

# Leistungskennzahlen
# Throughput
hist(Produktionsdaten$Throughput)

# Buffer1Mean
hist(Produktionsdaten$Buffer1Mean)

# Bufer2Mean
hist(Produktionsdaten$Buffer2Mean)

# CycleTimne
hist(Produktionsdaten$CycleTime)

# MachineXUtilization
# Machine1
hist(Produktionsdaten$Machine1Utilization)
# Machine2
hist(Produktionsdaten$Machine2Utilization)
# Machine3
hist(Produktionsdaten$Machine3Utilization)
# Machine4
hist(Produktionsdaten$Machine4Utilization)
```

Preprocessing

```
# WICHTIG: Library discretization, Library binr und die unten aufge-
führten Funktionen laden!
library(discretization)
library(binr)

setwd("C:/Users/User/Documents/Hauptseminar SVM")

# Equal-Frequency-Binning mit Package binr: gibt einen Vektor mit den
Gruppenwerten zurück
discretize <- function(matrix_column, target.bins, exact.groups) {
  matrix_column_bins <- bins(matrix_column, target.bins = target.bins,
exact.groups = exact.groups, minpts = 5000)
  matrix_column_bins_interval <- bins.getvals(matrix_column_bins)
  matrix_column_disc <- cut(matrix_column, breaks = ma-
trix_column_bins_interval, labels = FALSE, right = FALSE)
  return (matrix_column_disc)
}

# Normieren (/Skalieren) der Stellgrößen (Min-Max-Normalisierung)
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

#########################################################
# Laden des Datensatzes und Speichern als RDS-Datei (Data.frame) //
Korrektur des Spaltennamens
Produktionsdaten <- read.csv2(paste("./Data/", "Produktionsda-
ten2_80_400000.csv", sep = ""))
Produktionsdaten$CarrierSpeedEmpty <- Produktionsda-
ten$CarrierSpeedEmtpy
Produktionsdaten$CarrierSpeedEmtpy <- NULL
saveRDS(Produktionsdaten, "./Data/Produktionsdaten.rds")

# Lesen des Datensatzes
Produktionsdaten <- readRDS("./Data/Produktionsdaten.rds")

#########################################################
# Leistungskennzahlen
# Erstellen einer Matrix mit den Leistungskennzahlen
Leistungskennzahlen <- data.frame(cbind("Throughput" = Produktionsda-
ten$Throughput, "CycleTime" = Produktionsdaten$CycleTime, "Buffer1Mean"
= Produktionsdaten$Buffer1Mean, "Buffer2Mean" = Produktionsda-
ten$Buffer2Mean, "Buffer1Max" = Produktionsdaten$Buffer1Max, "Buf-
fer2Max" = Produktionsdaten$Buffer2Max, "Machine1Utilization" = Produk-
tionsdaten$Machine1Utilization, "Machine2Utilization" = Produktionsda-
```

```
ten$Machine2Utilization, "Machine3Utilization" = Produktionsda-
ten$Machine3Utilization, "Machine4Utilization" = Produktionsda-
ten$Machine4Utilization))
saveRDS(Leistungskennzahlen, "./Data/Leistungskennzahlen.rds")

# Verschieben des Kommas um 2 Stellen:
Leistungskennzahlen$Buffer1Mean <- Leistungskennzahlen$Buffer1Mean*100
Leistungskennzahlen$Buffer2Mean <- Leistungskennzahlen$Buffer2Mean*100
saveRDS(Leistungskennzahlen, "./Data/Leistungskennzahlen.rds")

Leistungskennzahlen <- readRDS("./Data/Leistungskennzahlen.rds")

# Diskretisieren
# Initialisierung eines leeren Dataframes
Leistungskennzahlen_diskretisiert <- data.frame(1:400000)
# Diskretisieren der Leistungskennzahlen in 5 Klassen
for(i in 1:length(Leistungskennzahlen)){
  print(paste("Diskretisieren von Leistungskennzahl", na-
mes(Leistungskennzahlen[i])))
  diskretisiert <- discretize(Leistungskennzahlen[[i]], 5, 5)
  diskretisiert_faktor <- factor(diskretisiert)
  Leistungskennzahlen_diskretisiert <-
cbind(Leistungskennzahlen_diskretisiert, diskretisiert_faktor)
}
Leistungskennzahlen_diskretisiert[1] <- NULL

# Namen der Spalten übernehmen
Leistungskennzahlen_diskretisiert <- setNa-
mes(Leistungskennzahlen_diskretisiert, c(names(Leistungskennzahlen)))

# Buffer1Max und Buffer2Max bestanden bereits aus diskreten Werten
Leistungskennzahlen_diskretisiert$Buffer1Max <- fac-
tor(Leistungskennzahlen$Buffer1Max)
Leistungskennzahlen_diskretisiert$Buffer2Max <- fac-
tor(Leistungskennzahlen$Buffer2Max)
saveRDS(Leistungskennzahlen_diskretisiert,
"./Data/Leistungskennzahlen_diskretisiert.rds")

############################################################
# Stellgrößen
# Erstellen einer Matrix mit den Stellgrößen
Stellgroessen <- data.frame(cbind("LoadingVolume" = Produktionsda-
ten$LoadingVolume, "Buffer1" = Produktionsdaten$Buffer1, "Buffer2" =
Produktionsdaten$Buffer2, "NumberOfCarriers" = Produktionsda-
ten$NumberOfCarriers, "CarrierSpeedLoaded" = Produktionsda-
ten$CarrierSpeedLoaded, "CarrierSpeedEmpty" = Produktionsda-
ten$CarrierSpeedEmpty, "Product1" = Produktionsdaten$Product1, "Pro-
```

```
duct2" = Produktionsdaten$Product2, "Product3" = Produktionsda-
ten$Product3, "Product4" = Produktionsdaten$Product4, "Product5" = Pro-
duktionsdaten$Product5, "Product6" = Produktionsdaten$Product6, "Pro-
duct7" = Produktionsdaten$Product7, "Product8" = Produktionsda-
ten$Product8, "Product9" = Produktionsdaten$Product9, "Product10" =
Produktionsdaten$Product10, "Product11" = Produktionsdaten$Product11,
"Product12" = Produktionsdaten$Product12, "Product13" = Produktionsda-
ten$Product13, "Product14" = Produktionsdaten$Product14, "Product15" =
Produktionsdaten$Product15, "Product16" = Produktionsdaten$Product16,
"Product17" = Produktionsdaten$Product17, "Product18" = Produktionsda-
ten$Product18, "Product19" = Produktionsdaten$Product19, "Product20" =
Produktionsdaten$Product20, "Product21" = Produktionsdaten$Product21,
"Product22" = Produktionsdaten$Product22, "Product23" = Produktionsda-
ten$Product23, "Product24" = Produktionsdaten$Product24, "Product25" =
Produktionsdaten$Product25, "Product26" = Produktionsdaten$Product26,
"Product27" = Produktionsdaten$Product27, "Machine1Speed" = Produkti-
onsdaten$Machine1Speed, "Machine2Speed" = Produktionsda-
ten$Machine2Speed, "Machine3Speed" = Produktionsdaten$Machine3Speed,
"Machine4Speed" = Produktionsdaten$Machine4Speed))
saveRDS(Stellgroessen, "./Data/Stellgroessen.rds")

# Normalisieren
Stellgroessen_normalisiert <- as.data.frame(lapply(Stellgroessen, nor-
malize))
saveRDS(Stellgroessen_normalisiert,
"./Data/Stellgroessen_normalisiert.rds")

############################################################
# Zusammenführen der Stellgrößen und der Leistungskennzahlen

# Daten für die Klassifikation
# Zusammenführen der normierten Stellgroessen und der diskretisierten
Leistungskennzahlen
Produktionsdaten_vorbearbeitet_Classification <- da-
ta.frame(Stellgroessen_normalisiert, Leistungskennzahlen_diskretisiert)
summary(Produktionsdaten_vorbearbeitet_Classification)
saveRDS(Produktionsdaten_vorbearbeitet_Classification,
"./Data/Produktionsdaten_vorbearbeitet_Classification.rds")

# Daten für die Regression
# Zusammenführen der normierten Stellgroessen und der Leistungskennzah-
len
Stellgroessen_normalisiert <-
readRDS("./Data/Stellgroessen_normalisiert.rds")
Leistungskennzahlen <- readRDS("./Data/Leistungskennzahlen.rds")
Produktionsdaten_vorbearbeitet_Regression <- da-
ta.frame(Stellgroessen_normalisiert, Leistungskennzahlen)
```

```
Produktionsdaten_vorbearbeitet_Regression$Buffer1Max <- NULL
Produktionsdaten_vorbearbeitet_Regression$Buffer2Max <- NULL
summary(Produktionsdaten_vorbearbeitet_Regression)
saveRDS(Produktionsdaten_vorbearbeitet_Regression,
"./Data/Produktionsdaten_vorbearbeitet_Regression.rds")
```

Classification

```
# WICHTIG: Library dplyr, Library e1071 und die unten aufgeführten
Funktionen laden!
library(dplyr)
library(e1071)

# Funktion zur Erstellung einer zufälliger Teilmenge
randomSample = function(df,n) {
  return (df[sample(nrow(df),n),])
}

# Validierung der SVC
validate.classification = function(model, testset, feature) {
  prediction <- predict(model, testset)
  cm = as.matrix(table(Actual = feature, Predicted = prediction))
  accuracy = (sum(diag(cm))/sum(cm))*100
  print(cm)
  print(paste("Accuracy: ", accuracy, "%"))
}

############################################################

# Einlesen der Daten
Produktionsdaten_vorbearbeitet_Classification <-
readRDS("./Data/Produktionsdaten_vorbearbeitet_Classification.rds")

# Erstellen eines Trainings- und Testdatensatzes
trainset <- randomSample(Produktionsdaten_vorbearbeitet_Classification,
2500)
testset <- randomSample(Produktionsdaten_vorbearbeitet_Classification,
50000)

# Listen für Tuning der SVC
cost_list <- c(10^-5, 10^-3, 10^-1, 10^1, 10^3, 10^5, 10^7)
gamma_list <- c(10^-7, 10^-5, 10^-3, 10^-1, 10^1, 10^3, 10^5)

############################################################
# Erstellen der SVC - sortiert nach Leistungskennzahl

# SVC für Buffer1Max
#SVC_Buffer1Max_tune <- tune(svm, Buffer1Max ~ LoadingVolume + Buffer1
+ Buffer2 + NumberOfCarriers + CarrierSpeedLoaded + CarrierSpeedEmpty +
Product1 + Product2 + Product3 + Product4 + Product5 + Product6 + Prod-
uct7 + Product8 + Product9 + Product10 + Product11 + Product12 + Prod-
uct13 + Product14 + Product15 + Product16 + Product17 + Product18 +
Product19 + Product20 + Product21 + Product22 + Product23 + Product24 +
```

```
Product25 + Product26 + Product27 + Machine1Speed + Machine2Speed +
Machine3Speed + Machine4Speed, data = trainset, type = "C-
classification", kernel = "radial", ranges = list(gamma = gamma_list,
cost = cost_list), scale = T)
#print(SVC_Buffer1Max_tune)
#plot(SVC_Buffer1Max_tune)
SVC_Buffer1Max <- svm(Buffer1Max ~ LoadingVolume + Buffer1 + Buffer2 +
NumberOfCarriers + CarrierSpeedLoaded + CarrierSpeedEmpty + Product1 +
Product2 + Product3 + Product4 + Product5 + Product6 + Product7 + Prod-
uct8 + Product9 + Product10 + Product11 + Product12 + Product13 + Prod-
uct14 + Product15 + Product16 + Product17 + Product18 + Product19 +
Product20 + Product21 + Product22 + Product23 + Product24 + Product25 +
Product26 + Product27 + Machine1Speed + Machine2Speed + Machine3Speed +
Machine4Speed, data = trainset, type = "C-classification", kernel =
"radial", cost = 100, gamma = 0.01, scale = T)
validate.classification(SVC_Buffer1Max, testset, testset$Buffer1Max)
saveRDS(SVC_Buffer1Max, "./Model/Classification/SVC_Buffer1Max.rds")

# SVC für Buffer2Max
#SVC_Buffer2Max_tune <- tune(svm, Buffer2Max ~ LoadingVolume + Buffer1
+ Buffer2 + NumberOfCarriers + CarrierSpeedLoaded + CarrierSpeedEmpty +
Product1 + Product2 + Product3 + Product4 + Product5 + Product6 + Prod-
uct7 + Product8 + Product9 + Product10 + Product11 + Product12 + Prod-
uct13 + Product14 + Product15 + Product16 + Product17 + Product18 +
Product19 + Product20 + Product21 + Product22 + Product23 + Product24 +
Product25 + Product26 + Product27 + Machine1Speed + Machine2Speed +
Machine3Speed + Machine4Speed, data = trainset, type = "C-
classification", kernel = "radial", ranges = list(gamma = gamma_list,
cost = cost_list), scale = T)
#print(SVC_Buffer2Max_tune)
#plot(SVC_Buffer2Max_tune)
SVC_Buffer2Max <- svm(Buffer2Max ~ LoadingVolume + Buffer1 + Buffer2 +
NumberOfCarriers + CarrierSpeedLoaded + CarrierSpeedEmpty + Product1 +
Product2 + Product3 + Product4 + Product5 + Product6 + Product7 + Prod-
uct8 + Product9 + Product10 + Product11 + Product12 + Product13 + Prod-
uct14 + Product15 + Product16 + Product17 + Product18 + Product19 +
Product20 + Product21 + Product22 + Product23 + Product24 + Product25 +
Product26 + Product27 + Machine1Speed + Machine2Speed + Machine3Speed +
Machine4Speed, data = trainset, type = "C-classification", kernel =
"radial", cost = 1000, gamma = 0.01, scale = T)
validate.classification(SVC_Buffer2Max, testset, testset$Buffer2Max)
saveRDS(SVC_Buffer2Max, "./Model/Classification/SVC_Buffer2Max.rds")

# SVC für Buffer1Mean
#SVC_Buffer1Mean_tune <- tune(svm, Buffer1Mean ~ LoadingVolume + Buff-
er1 + Buffer2 + NumberOfCarriers + CarrierSpeedLoaded + Carrier-
SpeedEmpty + Product1 + Product2 + Product3 + Product4 + Product5 +
```

```
Product6 + Product7 + Product8 + Product9 + Product10 + Product11 +
Product12 + Product13 + Product14 + Product15 + Product16 + Product17 +
Product18 + Product19 + Product20 + Product21 + Product22 + Product23 +
Product24 + Product25 + Product26 + Product27 + Machine1Speed + Ma-
chine2Speed + Machine3Speed + Machine4Speed, data = trainset, type =
"C-classification", kernel = "radial", ranges = list(gamma = gam-
ma_list, cost = cost_list), scale = T)
#print(SVC_Buffer1Mean_tune)
#plot(SVC_Buffer1Mean_tune)
SVC_Buffer1Mean <- svm(Buffer1Mean ~ LoadingVolume + Buffer1 + Buffer2
+ NumberOfCarriers + CarrierSpeedLoaded + CarrierSpeedEmpty + Product1
+ Product2 + Product3 + Product4 + Product5 + Product6 + Product7 +
Product8 + Product9 + Product10 + Product11 + Product12 + Product13 +
Product14 + Product15 + Product16 + Product17 + Product18 + Product19 +
Product20 + Product21 + Product22 + Product23 + Product24 + Product25 +
Product26 + Product27 + Machine1Speed + Machine2Speed + Machine3Speed +
Machine4Speed, data = trainset, type = "C-classification", kernel =
"radial", cost = 1000, gamma = 0.001, scale = T)
validate.classification(SVC_Buffer1Mean, testset, testset$Buffer1Mean)
saveRDS(SVC_Buffer1Mean, "./Model/Classification/SVC_Buffer1Mean.rds")

# SVC für Buffer2Mean
#SVC_Buffer2Mean_tune <- tune(svm, Buffer2Mean ~ LoadingVolume + Buff-
er1 + Buffer2 + NumberOfCarriers + CarrierSpeedLoaded + Carrier-
SpeedEmpty + Product1 + Product2 + Product3 + Product4 + Product5 +
Product6 + Product7 + Product8 + Product9 + Product10 + Product11 +
Product12 + Product13 + Product14 + Product15 + Product16 + Product17 +
Product18 + Product19 + Product20 + Product21 + Product22 + Product23 +
Product24 + Product25 + Product26 + Product27 + Machine1Speed + Ma-
chine2Speed + Machine3Speed + Machine4Speed, data = trainset, type =
"C-classification", kernel = "radial", ranges = list(gamma = gam-
ma_list, cost = cost_list), scale = T)
#print(SVC_Buffer2Mean_tune)
SVC_Buffer2Mean <- svm(Buffer2Mean ~ LoadingVolume + Buffer1 + Buffer2
+ NumberOfCarriers + CarrierSpeedLoaded + CarrierSpeedEmpty + Product1
+ Product2 + Product3 + Product4 + Product5 + Product6 + Product7 +
Product8 + Product9 + Product10 + Product11 + Product12 + Product13 +
Product14 + Product15 + Product16 + Product17 + Product18 + Product19 +
Product20 + Product21 + Product22 + Product23 + Product24 + Product25 +
Product26 + Product27 + Machine1Speed + Machine2Speed + Machine3Speed +
Machine4Speed, data = trainset, type = "C-classification", kernel =
"radial", cost = 1e+05, gamma = 0.001, scale = T)
validate.classification(SVC_Buffer2Mean, testset, testset$Buffer2Mean)
saveRDS(SVC_Buffer2Mean, "./Model/Classification/SVC_Buffer2Mean.rds")

# SVC für Throughput
```

```
#SVC_Throughput_tune <- tune(svm, Throughput ~ LoadingVolume + Buffer1
+ Buffer2 + NumberOfCarriers + CarrierSpeedLoaded + CarrierSpeedEmpty +
Product1 + Product2 + Product3 + Product4 + Product5 + Product6 + Prod-
uct7 + Product8 + Product9 + Product10 + Product11 + Product12 + Prod-
uct13 + Product14 + Product15 + Product16 + Product17 + Product18 +
Product19 + Product20 + Product21 + Product22 + Product23 + Product24 +
Product25 + Product26 + Product27 + Machine1Speed + Machine2Speed +
Machine3Speed + Machine4Speed, data = trainset, type = "C-
classification", kernel = "radial", ranges = list(gamma = c(0.0001,
0.001, 0.01, 0.1, 0.3, 0.5, 1), cost = c(1, 5, 10, 100, 500, 1000,
3000, 10000)), scale = T)
#plot(SVC_Throughput_tune)
#print(SVC_Throughput_tune)
SVC_Throughput <- svm(Throughput ~ LoadingVolume + Buffer1 + Buffer2 +
NumberOfCarriers + CarrierSpeedLoaded + CarrierSpeedEmpty + Product1 +
Product2 + Product3 + Product4 + Product5 + Product6 + Product7 + Prod-
uct8 + Product9 + Product10 + Product11 + Product12 + Product13 + Prod-
uct14 + Product15 + Product16 + Product17 + Product18 + Product19 +
Product20 + Product21 + Product22 + Product23 + Product24 + Product25 +
Product26 + Product27 + Machine1Speed + Machine2Speed + Machine3Speed +
Machine4Speed, data = trainset, type = "C-classification", kernel =
"radial", cost = 100, gamma = 0.01, scale = T)
validate.classification(SVC_Throughput, testset, testset$Throughput)
saveRDS(SVC_Throughput, "./Model/Classification/SVC_Throughput.rds")

# SVC für CycleTime
#SVC_CycleTime_tune <- tune(svm, CycleTime ~ LoadingVolume + Buffer1 +
Buffer2 + NumberOfCarriers + CarrierSpeedLoaded + CarrierSpeedEmpty +
Product1 + Product2 + Product3 + Product4 + Product5 + Product6 + Prod-
uct7 + Product8 + Product9 + Product10 + Product11 + Product12 + Prod-
uct13 + Product14 + Product15 + Product16 + Product17 + Product18 +
Product19 + Product20 + Product21 + Product22 + Product23 + Product24 +
Product25 + Product26 + Product27 + Machine1Speed + Machine2Speed +
Machine3Speed + Machine4Speed, data = trainset, type = "C-
classification", kernel = "radial", ranges = list(gamma = c(0.0001,
0.001, 0.01, 0.1, 0.3, 0.5, 1), cost = c(1, 5, 10, 100, 500, 1000,
3000, 10000)), scale = T)
#plot(SVC_CycleTime_tune)
#print(SVC_CycleTime_tune)
#saveRDS(SVC_CycleTime_tune, "./CycleTime_tune.rds")
SVC_CycleTime <- svm(CycleTime ~ LoadingVolume + Buffer1 + Buffer2 +
NumberOfCarriers + CarrierSpeedLoaded + CarrierSpeedEmpty + Product1 +
Product2 + Product3 + Product4 + Product5 + Product6 + Product7 + Prod-
uct8 + Product9 + Product10 + Product11 + Product12 + Product13 + Prod-
uct14 + Product15 + Product16 + Product17 + Product18 + Product19 +
Product20 + Product21 + Product22 + Product23 + Product24 + Product25 +
Product26 + Product27 + Machine1Speed + Machine2Speed + Machine3Speed +
```

```
Machine4Speed, data = trainset, type = "C-classification", kernel =
"radial", cost = 10000, gamma = 0.001, scale = T)
validate.classification(SVC_CycleTime, testset, testset$CycleTime)
saveRDS(SVC_CycleTime, "./Model/Classification/SVC_CycleTime.rds")

# SVC für Machine1Utilization
#SVC_Machine1Utilization_tune <- tune(svm, Machine1Utilization ~ Load-
ingVolume + Buffer1 + Buffer2 + NumberOfCarriers + CarrierSpeedLoaded +
CarrierSpeedEmpty + Product1 + Product2 + Product3 + Product4 + Prod-
uct5 + Product6 + Product7 + Product8 + Product9 + Product10 + Prod-
uct11 + Product12 + Product13 + Product14 + Product15 + Product16 +
Product17 + Product18 + Product19 + Product20 + Product21 + Product22 +
Product23 + Product24 + Product25 + Product26 + Product27 + Ma-
chine1Speed + Machine2Speed + Machine3Speed + Machine4Speed, data =
trainset, type = "C-classification", kernel = "radial", ranges =
list(gamma = c(0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 1), cost = c(1, 5,
10, 100, 500, 1000, 3000, 10000)), scale = T)
#plot(SVC_Machine1Utilization_tune)
#print(SVC_Machine1Utilization_tune)
SVC_Machine1Utilization <- svm(Machine1Utilization ~ LoadingVolume +
Buffer1 + Buffer2 + NumberOfCarriers + CarrierSpeedLoaded + Carrier-
SpeedEmpty + Product1 + Product2 + Product3 + Product4 + Product5 +
Product6 + Product7 + Product8 + Product9 + Product10 + Product11 +
Product12 + Product13 + Product14 + Product15 + Product16 + Product17 +
Product18 + Product19 + Product20 + Product21 + Product22 + Product23 +
Product24 + Product25 + Product26 + Product27 + Machine1Speed + Ma-
chine2Speed + Machine3Speed + Machine4Speed, data = trainset, type =
"C-classification", kernel = "radial", cost = 2, gamma = 0.1, scale =
T)
validate.classification(SVC_Machine1Utilization, testset, test-
set$Machine1Utilization)
saveRDS(SVC_Machine1Utilization,
"./Model/Classification/SVC_Machine1Utilization.rds")

# SVC für Machine2Utilization
#SVC_Machine2Utilization_tune <- tune(svm, Machine2Utilization ~ Load-
ingVolume + Buffer1 + Buffer2 + NumberOfCarriers + CarrierSpeedLoaded +
CarrierSpeedEmpty + Product1 + Product2 + Product3 + Product4 + Prod-
uct5 + Product6 + Product7 + Product8 + Product9 + Product10 + Prod-
uct11 + Product12 + Product13 + Product14 + Product15 + Product16 +
Product17 + Product18 + Product19 + Product20 + Product21 + Product22 +
Product23 + Product24 + Product25 + Product26 + Product27 + Ma-
chine1Speed + Machine2Speed + Machine3Speed + Machine4Speed, data =
trainset, type = "C-classification", kernel = "radial", ranges =
list(gamma = c(0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 1), cost = c(1, 5,
10, 100, 500, 1000, 3000, 10000)), scale = T)
#plot(SVC_Machine2Utilization_tune)
```

```
#print(SVC_Machine2Utilization_tune)
SVC_Machine2Utilization <- svm(Machine2Utilization ~ LoadingVolume +
Buffer1 + Buffer2 + NumberOfCarriers + CarrierSpeedLoaded + Carrier-
SpeedEmpty + Product1 + Product2 + Product3 + Product4 + Product5 +
Product6 + Product7 + Product8 + Product9 + Product10 + Product11 +
Product12 + Product13 + Product14 + Product15 + Product16 + Product17 +
Product18 + Product19 + Product20 + Product21 + Product22 + Product23 +
Product24 + Product25 + Product26 + Product27 + Machine1Speed + Ma-
chine2Speed + Machine3Speed + Machine4Speed, data = trainset, type =
"C-classification", kernel = "radial", cost = 1000, gamma = 0.01, scale
= T)
validate.classification(SVC_Machine2Utilization, testset, test-
set$Machine2Utilization)
saveRDS(SVC_Machine2Utilization,
"./Model/Classification/SVC_Machine2Utilization.rds")

# SVM für Machine3Utilization
#SVC_Machine3Utilization_tune <- tune(svm, Machine3Utilization ~ Load-
ingVolume + Buffer1 + Buffer2 + NumberOfCarriers + CarrierSpeedLoaded +
CarrierSpeedEmpty + Product1 + Product2 + Product3 + Product4 + Prod-
uct5 + Product6 + Product7 + Product8 + Product9 + Product10 + Prod-
uct11 + Product12 + Product13 + Product14 + Product15 + Product16 +
Product17 + Product18 + Product19 + Product20 + Product21 + Product22 +
Product23 + Product24 + Product25 + Product26 + Product27 + Ma-
chine1Speed + Machine2Speed + Machine3Speed + Machine4Speed, data =
trainset, type = "C-classification", kernel = "radial", ranges =
list(gamma = c(0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 1), cost = c(1, 5,
10, 100, 500, 1000, 3000, 10000)), scale = T)
#plot(SVC_Machine3Utilization_tune)
#print(SVC_Machine3Utilization_tune)
SVC_Machine3Utilization <- svm(Machine3Utilization ~ LoadingVolume +
Buffer1 + Buffer2 + NumberOfCarriers + CarrierSpeedLoaded + Carrier-
SpeedEmpty + Product1 + Product2 + Product3 + Product4 + Product5 +
Product6 + Product7 + Product8 + Product9 + Product10 + Product11 +
Product12 + Product13 + Product14 + Product15 + Product16 + Product17 +
Product18 + Product19 + Product20 + Product21 + Product22 + Product23 +
Product24 + Product25 + Product26 + Product27 + Machine1Speed + Ma-
chine2Speed + Machine3Speed + Machine4Speed, data = trainset, type =
"C-classification", kernel = "radial", cost = 1000, gamma = 0.01, scale
= T)
validate.classification(SVC_Machine3Utilization, testset, test-
set$Machine3Utilization)
saveRDS(SVC_Machine3Utilization,
"./Model/Classification/SVC_Machine3Utilization.rds")

# SVM für Machine4Utilization
```

```
#SVC_Machine4Utilization_tune <- tune(svm, Machine1Utilization ~ Load-
ingVolume + Buffer1 + Buffer2 + NumberOfCarriers + CarrierSpeedLoaded +
CarrierSpeedEmpty + Product1 + Product2 + Product3 + Product4 + Prod-
uct5 + Product6 + Product7 + Product8 + Product9 + Product10 + Prod-
uct11 + Product12 + Product13 + Product14 + Product15 + Product16 +
Product17 + Product18 + Product19 + Product20 + Product21 + Product22 +
Product23 + Product24 + Product25 + Product26 + Product27 + Ma-
chine1Speed + Machine2Speed + Machine3Speed + Machine4Speed, data =
trainset, type = "C-classification", kernel = "radial", ranges =
list(gamma = c(0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 1), cost = c(1, 5,
10, 100, 500, 1000, 3000, 10000)), scale = T)
#plot(SVC_Machine4Utilization_tune)
#print(SVC_Machine4Utilization_tune)
SVC_Machine4Utilization <- svm(Machine4Utilization ~ LoadingVolume +
Buffer1 + Buffer2 + NumberOfCarriers + CarrierSpeedLoaded + Carrier-
SpeedEmpty + Product1 + Product2 + Product3 + Product4 + Product5 +
Product6 + Product7 + Product8 + Product9 + Product10 + Product11 +
Product12 + Product13 + Product14 + Product15 + Product16 + Product17 +
Product18 + Product19 + Product20 + Product21 + Product22 + Product23 +
Product24 + Product25 + Product26 + Product27 + Machine1Speed + Ma-
chine2Speed + Machine3Speed + Machine4Speed, data = trainset, type =
"C-classification", kernel = "radial", cost = 1000, gamma = 0.01, scale
= T)
validate.classification(SVC_Machine4Utilization, testset, test-
set$Machine4Utilization)
saveRDS(SVC_Machine4Utilization,
"./Model/Classification/SVC_Machine4Utilization.rds")
```

Regression

```
# WICHTIG: Library e1071 und die unten aufgeführten Funktionen laden!
library(e1071)

# Funktion zur Erstellung einer zufälliger Teilmenge
randomSample = function(df,n) {
  return (df[sample(nrow(df),n),])
}

# Validierung der SVR
validate.regression = function(model, testset, feature) {
  prediction <- predict(model, testset)
  error_norm = abs(feature-prediction)/feature
  error_norm_max = max(error_norm)
  error_norm_min = min(error_norm)
  error_norm_mean = mean(error_norm)
  print(paste("Normierte absolute Standardabweichung: ", er-
ror_norm_mean))
  print(paste("Maximale normierte Abweichung: ", error_norm_max, " Mi-
nimale Abweichung: ", error_norm_min))
  return(list("error_norm" = error_norm, "error_norm_mean" = er-
ror_norm_mean, "error_max" = error_norm_max, "error_min" = er-
ror_norm_min))
}

############################################################

Produktionsdaten_vorbearbeitet_Regression <-
readRDS("./Data/Produktionsdaten_vorbearbeitet_Regression.rds")

trainset <- randomSample(Produktionsdaten_vorbearbeitet_Regression,
10000)
testset <- randomSample(Produktionsdaten_vorbearbeitet_Regression,
400000)

# Listen für Tuning
cost_list <- c(10^-5, 10^-3, 10^-1, 10^1, 10^3, 10^5, 10^7)
gamma_list <- c(10^-7, 10^-5, 10^-3, 10^-1, 10^1, 10^3, 10^5)

############################################################
# Erstellen der SVR - sortiert nach Leistungskennzahl

# SVR für Throughput
#SVR_Throughput_tune <- tune(svm, Throughput ~ LoadingVolume + Buffer1
+ Buffer2 + NumberOfCarriers + CarrierSpeedLoaded + CarrierSpeedEmpty +
```

```
Product1 + Product2 + Product3 + Product4 + Product5 + Product6 + Prod-
uct7 + Product8 + Product9 + Product10 + Product11 + Product12 + Prod-
uct13 + Product14 + Product15 + Product16 + Product17 + Product18 +
Product19 + Product20 + Product21 + Product22 + Product23 + Product24 +
Product25 + Product26 + Product27 + Machine1Speed + Machine2Speed +
Machine3Speed + Machine4Speed, data = trainset, type = "eps-
regression", kernel = "radial", ranges = list(gamma = gamma_list, cost
= cost_list), scale = T)
#print(SVR_Throughput_tune)
#plot(SVR_Throughput_tune)
SVR_Throughput <- svm(Throughput ~ LoadingVolume + Buffer1 + Buffer2 +
NumberOfCarriers + CarrierSpeedLoaded + CarrierSpeedEmpty + Product1 +
Product2 + Product3 + Product4 + Product5 + Product6 + Product7 + Prod-
uct8 + Product9 + Product10 + Product11 + Product12 + Product13 + Prod-
uct14 + Product15 + Product16 + Product17 + Product18 + Product19 +
Product20 + Product21 + Product22 + Product23 + Product24 + Product25 +
Product26 + Product27 + Machine1Speed + Machine2Speed + Machine3Speed +
Machine4Speed , data = trainset)
SVR_Throughput_validation <- validate.regression(SVR_Throughput, test-
set, testset$Throughput)
hist(SVR_Throughput_validation$error_norm, breaks = seq(0,
max(SVR_Throughput_validation$error_norm)+0.05, 0.05), main = "Histo-
gramm der normierten Abweichung - Throughput", xlab = "normierte Abwei-
chung", ylab = "Häufigkeit")
saveRDS(SVR_Throughput, "./Model/Regression/SVR_Throughput.rds")

# SVR für Cycletime
SVR_CycleTime <- svm(CycleTime ~ LoadingVolume + Buffer1 + Buffer2 +
NumberOfCarriers + CarrierSpeedLoaded + CarrierSpeedEmpty + Product1 +
Product2 + Product3 + Product4 + Product5 + Product6 + Product7 + Prod-
uct8 + Product9 + Product10 + Product11 + Product12 + Product13 + Prod-
uct14 + Product15 + Product16 + Product17 + Product18 + Product19 +
Product20 + Product21 + Product22 + Product23 + Product24 + Product25 +
Product26 + Product27 + Machine1Speed + Machine2Speed + Machine3Speed +
Machine4Speed , data = trainset)
SVR_CycleTime_validation <- validate.regression(SVR_CycleTime, testset,
testset$CycleTime)
hist(SVR_CycleTime_validation$error_norm, breaks = seq(0,
max(SVR_CycleTime_validation$error_norm)+0.05, 0.05), main = "Histo-
gramm der normierten Abweichung - CycleTime", xlab = "normierte Abwei-
chung", ylab = "Häufigkeit")
saveRDS(SVR_CycleTime, "./Model/Regression/SVR_CycleTime.rds")

# SVR für Buffer1Mean
SVR_Buffer1Mean <- svm(Buffer1Mean ~ LoadingVolume + Buffer1 + Buffer2
+ NumberOfCarriers + CarrierSpeedLoaded + CarrierSpeedEmpty + Product1
+ Product2 + Product3 + Product4 + Product5 + Product6 + Product7 +
```

```
Product8 + Product9 + Product10 + Product11 + Product12 + Product13 +
Product14 + Product15 + Product16 + Product17 + Product18 + Product19 +
Product20 + Product21 + Product22 + Product23 + Product24 + Product25 +
Product26 + Product27 + Machine1Speed + Machine2Speed + Machine3Speed +
Machine4Speed , data = trainset)
SVR_Buffer1Mean_validation <- validate.regression(SVR_Buffer1Mean,
testset, testset$Buffer1Mean)
hist(SVR_Buffer1Mean_validation$error_norm, breaks = seq(0,
max(SVR_Buffer1Mean_validation$error_norm)+0.05, 0.05), main = "Histo-
gramm der normierten Abweichung - Buffer1Mean", xlab = "normierte Ab-
weichung", ylab = "Häufigkeit")
saveRDS(SVR_Buffer1Mean, "./Model/Regression/SVR_Buffer1Mean.rds")

# SVR für Buffer2Mean
SVR_Buffer2Mean <- svm(Buffer2Mean ~ LoadingVolume + Buffer1 + Buffer2
+ NumberOfCarriers + CarrierSpeedLoaded + CarrierSpeedEmpty + Product1
+ Product2 + Product3 + Product4 + Product5 + Product6 + Product7 +
Product8 + Product9 + Product10 + Product11 + Product12 + Product13 +
Product14 + Product15 + Product16 + Product17 + Product18 + Product19 +
Product20 + Product21 + Product22 + Product23 + Product24 + Product25 +
Product26 + Product27 + Machine1Speed + Machine2Speed + Machine3Speed +
Machine4Speed , data = trainset)
SVR_Buffer2Mean_validation <- validate.regression(SVR_Buffer2Mean,
testset, testset$Buffer2Mean)
hist(SVR_Buffer2Mean_validation$error_norm, breaks = seq(0,
max(SVR_Buffer2Mean_validation$error_norm)+0.05, 0.05), main = "Histo-
gramm der normierten Abweichung - Buffer2Mean", xlab = "normierte Ab-
weichung", ylab = "Häufigkeit")
saveRDS(SVR_Buffer2Mean, "./Model/Regression/SVR_Buffer2Mean.rds")

# SVR für Machine1Utilization
SVR_Machine1Utilization <- svm(Machine1Utilization ~ LoadingVolume +
Buffer1 + Buffer2 + NumberOfCarriers + CarrierSpeedLoaded + Carrier-
SpeedEmpty + Product1 + Product2 + Product3 + Product4 + Product5 +
Product6 + Product7 + Product8 + Product9 + Product10 + Product11 +
Product12 + Product13 + Product14 + Product15 + Product16 + Product17 +
Product18 + Product19 + Product20 + Product21 + Product22 + Product23 +
Product24 + Product25 + Product26 + Product27 + Machine1Speed + Ma-
chine2Speed + Machine3Speed + Machine4Speed , data = trainset)
SVR_Machine1Utilization_validation <- vali-
date.regression(SVR_Machine1Utilization, testset, test-
set$Machine1Utilization)
hist(SVR_Machine1Utilization_validation$error_norm, breaks = seq(0,
max(SVR_Machine1Utilization_validation$error_norm)+0.05, 0.05), main =
"Histogramm der normierten Abweichung - Machine1Utilization", xlab =
"normierte Abweichung", ylab = "Häufigkeit")
```

```
saveRDS(SVR_Machine1Utilization,
"./Model/Regression/SVR_Machine1Utilization.rds")

# SVR für Machine2Utilization
SVR_Machine2Utilization <- svm(Machine2Utilization ~ LoadingVolume +
Buffer1 + Buffer2 + NumberOfCarriers + CarrierSpeedLoaded + Carrier-
SpeedEmpty + Product1 + Product2 + Product3 + Product4 + Product5 +
Product6 + Product7 + Product8 + Product9 + Product10 + Product11 +
Product12 + Product13 + Product14 + Product15 + Product16 + Product17 +
Product18 + Product19 + Product20 + Product21 + Product22 + Product23 +
Product24 + Product25 + Product26 + Product27 + Machine1Speed + Ma-
chine2Speed + Machine3Speed + Machine4Speed , data = trainset)
SVR_Machine2Utilization_validation <- vali-
date.regression(SVR_Machine2Utilization, testset, test-
set$Machine2Utilization)
hist(SVR_Machine2Utilization_validation$error_norm, breaks = seq(0,
max(SVR_Machine2Utilization_validation$error_norm)+0.05, 0.05), main =
"Histogramm der normierten Abweichung - Machine2Utilization", xlab =
"normierte Abweichung", ylab = "Häufigkeit")
saveRDS(SVR_Machine2Utilization,
"./Model/Regression/SVR_Machine2Utilization.rds")

# SVR für Machine3Utilization
SVR_Machine3Utilization <- svm(Machine3Utilization ~ LoadingVolume +
Buffer1 + Buffer2 + NumberOfCarriers + CarrierSpeedLoaded + Carrier-
SpeedEmpty + Product1 + Product2 + Product3 + Product4 + Product5 +
Product6 + Product7 + Product8 + Product9 + Product10 + Product11 +
Product12 + Product13 + Product14 + Product15 + Product16 + Product17 +
Product18 + Product19 + Product20 + Product21 + Product22 + Product23 +
Product24 + Product25 + Product26 + Product27 + Machine1Speed + Ma-
chine2Speed + Machine3Speed + Machine4Speed , data = trainset)
SVR_Machine3Utilization_validation <- vali-
date.regression(SVR_Machine3Utilization, testset, test-
set$Machine3Utilization)
hist(SVR_Machine3Utilization_validation$error_norm, breaks = seq(0,
max(SVR_Machine3Utilization_validation$error_norm)+0.05, 0.05), main =
"Histogramm der normierten Abweichung - Machine3Utilization", xlab =
"normierte Abweichung", ylab = "Häufigkeit")
saveRDS(SVR_Machine3Utilization,
"./Model/Regression/SVR_Machine3Utilization.rds")

# SVR für Machine4Utilization
SVR_Machine4Utilization <- svm(Machine4Utilization ~ LoadingVolume +
Buffer1 + Buffer2 + NumberOfCarriers + CarrierSpeedLoaded + Carrier-
SpeedEmpty + Product1 + Product2 + Product3 + Product4 + Product5 +
Product6 + Product7 + Product8 + Product9 + Product10 + Product11 +
Product12 + Product13 + Product14 + Product15 + Product16 + Product17 +
```

```
Product18 + Product19 + Product20 + Product21 + Product22 + Product23 +
Product24 + Product25 + Product26 + Product27 + Machine1Speed + Ma-
chine2Speed + Machine3Speed + Machine4Speed , data = trainset)
SVR_Machine4Utilization_validation <- vali-
date.regression(SVR_Machine4Utilization, testset, test-
set$Machine4Utilization)
hist(SVR_Machine4Utilization_validation$error_norm, breaks = seq(0,
max(SVR_Machine4Utilization_validation$error_norm)+0.05, 0.05), main =
"Histogramm der normierten Abweichung - Machine4Utilization", xlab =
"normierte Abweichung", ylab = "Häufigkeit")
saveRDS(SVR_Machine4Utilization,
"./Model/Regression/SVR_Machine4Utilization.rds")
```

Application

```
# WICHTIG: Library e1071 und die unten aufgeführten Funktionen laden!
library(e1071)

# Preprocessing
preprocessing <- function(path, name){
  # Einlesen der csv-Datei und Abspeichern als rds-Datei
  Testdaten <- read.csv2(paste(path, name, sep = ""))
  saveRDS(Testdaten, paste(path, name, ".rds", sep = ""))

  # Normalisieren (/Skalieren) der Stellgrößen (Min-Max-Normalisierung)
  normalize <- function(x) {
    return ((x - min(x)) / (max(x) - min(x)))
  }

  Testdaten_normalisiert <- as.data.frame(lapply(Testdaten, normalize))
  return(Testdaten_normalisiert)
}

# Regression
svm.regression <- function(features, path, dataset){
  for(i in features){
    print(paste("Regression des Features", i))
    model <- readRDS(paste(path, "SVR_", i, ".rds", sep = ""))
    prediction <- predict(model, dataset)
    dataset <- data.frame(cbind(dataset, prediction))
    colnames(dataset)[length(colnames(dataset))] <- paste("Prediction",
i)
  }
  return(dataset)
}

# Klassifikation
svm.classification <- function(features, path, dataset){
  for(i in features){
    print(paste("Klassifikation des Features", i))
    model <- readRDS(paste(path, "SVC_", i, ".rds", sep = ""))
    prediction <- predict(model, dataset)
    dataset <- data.frame(cbind(dataset, name = prediction))
    colnames(dataset)[length(colnames(dataset))] <- paste("Prediction",
i)
  }
  return(dataset)
}

############################################################
```

```
# Dateinamen und Verzeichnisse eingeben

# Datensatz, dessen Leistungskennzahlen vorhergesagt werden sollen
dataset <- "Produktionsdaten2_80_400000.csv"

# Verzeichnisse des Datensatzes und der Modelle
datapath <- "./Data/"
modelpath_regression <- "./Model/Regression/"
modelpath_classification <- "./Model/Classification/"

# Vorherzusagende Leistungskennzahlen
features_regression = c("Throughput", "CycleTime", "Buffer1Mean",
"Buffer2Mean", "Machine1Utilization", "Machine2Utilization", "Ma-
chine3Utilization", "Machine4Utilization")
features_classification = c("Throughput", "CycleTime", "Buffer1Mean",
"Buffer1Max", "Buffer2Mean", "Buffer2Max", "Machine1Utilization", "Ma-
chine2Utilization", "Machine3Utilization", "Machine4Utilization")

############################################################
# Vorbereitung der Stellgrößen
Testdaten_preprocessed <- preprocessing(datapath, dataset)
Testdaten_preprocessed$CarrierSpeedEmpty <- Test-
daten_preprocessed$CarrierSpeedEmtpy
Testdaten_preprocessed$CarrierSpeedEmtpy <- NULL

############################################################
# Anwendung der Modelle auf den Datensatz
# Aufühen der Regression
result_regression <- svm.regression(features = features_regression,
path = modelpath_regression, dataset = Testdaten_preprocessed)
# Abspeichern als R-Datei und csv
saveRDS(result_regression, "./Data/Ergebnis_Regression.rds")
write.csv2(result_regression, "./Data/Ergebnis_Regression.csv")

# Ausführen der Klassifikation
result_classification <- svm.classification(features = fea-
tures_classification, path = modelpath_classification, dataset = Test-
daten_preprocessed)
# Abspeichern als R-Datei und csv
write.csv2(result_classification, "./Data/Ergebnis_Classification.csv")
saveRDS(result_classification, "./Data/Ergebnis_Classification.rds")
```

Evaluation

```
# WICHTIG: Library e1071, Library grDevices und die unten aufgeführten
Funktionen laden!

library(e1071)
library(grDevices)
library(binr)

evaluation.regression = function(feature){
  dataset = Produktionsdaten_vorbearbeitet_Regression
  feature_name = colnames(dataset)[feature]
  model <- get(paste("SVR_", feature_name, sep = ""))
  kernel <- model$kernel
  cost <- model$cost
  gamma <- model$gamma
  prediction <- predict(model, dataset)
  actual <- dataset[[feature]]
  error_norm = abs(actual-prediction)/actual
  error_norm_max = max(error_norm)
  error_norm_min = min(error_norm)
  error_norm_mean = mean(error_norm)

  # Histogramm
  png(filename= paste("./Graphics/Histogram/", feature_name, ".png",
sep = ""), width = 647, height = 496, units = "px")
  plot(hist(error_norm, breaks = seq(0, max(error_norm)+0.05, 0.05)),
main = "Histogramm der normierten Abweichung", sub = feature_name, xlab
= "normierte Abweichung", ylab = "Häufigkeit")
  dev.off()
  return(list("Leistungskennzahl" = feature_name, "Mittlere normierte
Abweichung" = error_norm_mean, "Maximale normierte Abweichung" = er-
ror_norm_max, "Minimale normierte Abweichung" = error_norm_min,
"Kernel" = kernel, "Kostenbudget" = cost, "Gamma" = gamma))
}

evaluation.classification = function(feature){
  dataset = Produktionsdaten_vorbearbeitet_Classification
  feature_name = colnames(dataset)[feature]
  model <- get(paste("SVC_", feature_name, sep = ""))
  kernel <- model$kernel
  cost <- model$cost
  gamma <- model$gamma
  prediction <- predict(model, dataset)
  actual <- dataset[feature]
  cm = as.matrix(table(Actual = actual[[1]], Predicted = prediction))
  accuracy = (sum(diag(cm))/sum(cm))*100
```

```
  return(list("Leistungskennzahl" = as.character(feature_name), "Genau-
igkeit" = accuracy, "Kernel" = kernel, "Kostenbudget" = cost, "Gamma" =
gamma))
}

############################################################
# Evaluation der Regressionsmodelle
# Einlesen der Daten
Produktionsdaten_vorbearbeitet_Regression <-
readRDS("./Data/Produktionsdaten_vorbearbeitet_Regression.rds")

# Einlesen der SVR-Modelle
SVR_Throughput <- readRDS("./Model/Regression/SVR_Throughput.rds")
SVR_CycleTime <- readRDS("./Model/Regression/SVR_CycleTime.rds")
SVR_Buffer1Mean <- readRDS("./Model/Regression/SVR_Buffer1Mean.rds")
SVR_Buffer2Mean <- readRDS("./Model/Regression/SVR_Buffer2Mean.rds")
SVR_Machine1Utilization <-
readRDS("./Model/Regression/SVR_Machine1Utilization.rds")
SVR_Machine2Utilization <-
readRDS("./Model/Regression/SVR_Machine2Utilization.rds")
SVR_Machine3Utilization <-
readRDS("./Model/Regression/SVR_Machine3Utilization.rds")
SVR_Machine4Utilization <-
readRDS("./Model/Regression/SVR_Machine4Utilization.rds")

# Anwenden des Evaluationsalogrithmus auf alle Modelle, die Ergebnisse
werden als Spalten angehangen
evaluation_regression <- data.frame(cbind("Leistungskennzahl" = 0,
"Mittlere normierte Abweichung" = 0, "Maximale normierte Abweichung" =
0, "Minimale normierte Abweichung" = 0, "Kernel" = 0, "Kostenbudget" =
0, "Gamma" = 0))
for(i in 38:45){
  evaluation_regression <- data.frame(rbind(evaluation_regression,
evaluation.regression(i)))
}

# Speichern der Evaluation als rds-Datei
saveRDS(evaluation_regression,
"./Model/Evaluation/evaluation_regression.rds")

############################################################
# Evaluation der Klassifikationsmodelle
# Einlesen der Daten
Produktionsdaten_vorbearbeitet_Classification <-
readRDS("./Data/Produktionsdaten_vorbearbeitet_Classification.rds")

# Einlesen der SVC-Modelle
```

```
SVC_Throughput <- readRDS("./Model/Classification/SVC_Throughput.rds")
SVC_CycleTime <- readRDS("./Model/Classification/SVC_CycleTime.rds")
SVC_Buffer1Mean <-
readRDS("./Model/Classification/SVC_Buffer1Mean.rds")
SVC_Buffer2Mean <-
readRDS("./Model/Classification/SVC_Buffer2Mean.rds")
SVC_Buffer1Max <- readRDS("./Model/Classification/SVC_Buffer1Max.rds")
SVC_Buffer2Max <- readRDS("./Model/Classification/SVC_Buffer2Max.rds")
SVC_Machine1Utilization <-
readRDS("./Model/Classification/SVC_Machine1Utilization.rds")
SVC_Machine2Utilization <-
readRDS("./Model/Classification/SVC_Machine2Utilization.rds")
SVC_Machine3Utilization <-
readRDS("./Model/Classification/SVC_Machine3Utilization.rds")
SVC_Machine4Utilization <-
readRDS("./Model/Classification/SVC_Machine4Utilization.rds")

# Anwenden des Evaluationsalogrithmus auf alle Modelle, die Ergebnisse
werden als Spalten angehangen
evaluation_classification <- data.frame(cbind("Leistungskennzahl" = 0,
"Genauigkeit" = 0, "Kernel" = 0, "Kostenbudget" = 0, "Gamma" = 0))
for(i in 38:47){
  evaluation_classification <- da-
ta.frame(rbind(evaluation_classification, evalua-
tion.classification(i)))
}

# Speichern der Evaluation als rds-Datei
saveRDS(evaluation_classification,
"./Model/Evaluation/evaluation_classifcation.rds")

################################################################
# Erstellen eines Balkendiagrammes zur Visualisierung von normierter
mittlerer absoluter Abweichung und normierter maximaler absoluter Ab-
weichung

evaluation_regression <-
readRDS("./Model/Evaluation/evaluation_regression.rds")
evaluation_classification <-
readRDS("./Model/Evaluation/evaluation_classifcation.rds")

# Erstellen eines Balkendiagramms
bartable <- data.frame(NULL)
feature <- rapply(evaluation_regression$feature, c)
error_norm_mean <- rapply((evaluation_regression$error_norm_mean), c)
error_norm_max <- rapply((evaluation_regression$error_max), c)
```

```
bartable <- data.frame("feature" = feature, "error_mean" = er-
ror_norm_mean, "error_max" = error_norm_max)

funProp <- function(testCol) {
  bartable[, testCol]/max(bartable[, testCol])
}

bartable$var.a.prop <- funProp(2)
bartable$var.b.prop <- funProp(3)
par(cex = 1.5, cex.main = 0.8)
barplot(t(as.matrix(bartable[, c("var.a.prop", "var.b.prop")])), beside
= TRUE,
        yaxt = "n", names.arg = test$feature, cex.names = 0.5, las = 2,
col = c("grey", "grey20"), main = "Evaluation der SVR")
par(cex = 0.6)
axis(2, line = 0, at = seq(0, max(bartable$var.a.prop), length.out =
11),
     labels = round(seq(0, max(bartable$error_mean), length.out = 11),
digits = 2), las =2)

axis(4, line = 0, at = seq(0, max(bartable$var.b.prop), length.out =
11),
     labels = round(seq(0, 4, length.out = 11), digits = 2), las = 2)

legend("topright",
       legend = c("mittlere Abweichung", "maximale Abweichung"),
       fill = c("grey", "grey20"), bty = "n")
mtext("mittlere normierte Abweichung",side=2,line=1.7, cex = 0.7)
mtext("maximale normierte Abweichung",side=4,line=1.2, cex = 0.7)

################################################################
validate.regression.cm = function(actual, prediction) {
  cm = as.matrix(table(Actual = actual, Predicted = prediction))
  accuracy = (sum(diag(cm))/sum(cm))*100
  print(cm)
  print(paste("Accuracy: ", accuracy, "%"))
}

# Vergleichen von Regression und Klassifikation
Leistungskennzahlen_diskretisiert <-
readRDS("./Data/Leistungskennzahlen_diskretisiert.rds")
Ergebnis_Regression <- readRDS("./Data/Ergebnis_Regression.rds")
Leistungskennzahlen <- readRDS("./Data/Leistungskennzahlen.rds")
# bei folgenden Leistungskennzahlen wurde keine Regression durchgeführt
Leistungskennzahlen$Buffer1Max <- NULL
Leistungskennzahlen$Buffer2Max <- NULL
```

```
for(i in 49:56){
  # Breakpoints laden
  matrix_column_bins <- bins(Leistungskennzahlen[[(i-48)]], target.bins
= 5, exact.groups = 5, minpts = 5000)
  matrix_column_bins_interval <- bins.getvals(matrix_column_bins)
  x <- cut(Ergebnis_Regression[[i]], breaks = ma-
trix_column_bins_interval, labels = FALSE, right = FALSE)
  validate.regression.cm(Leistungskennzahlen_diskretisiert[[i-47]], x)
}
```