Lehrstuhl für Umformtechnik

Masterarbeit

Entwicklung einer Support Vector Machine zur Eventklassifizierung und -detektion im Bergbau

Sabit Taygun Özdemir, BSc

November 2022

MONTANUNIVERSITÄT LEOBEN
www.unileoben.ac.at

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt, und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Ich erkläre, dass ich die Richtlinien des Senats der Montanuniversität Leoben zu "Gute wissenschaftliche Praxis" gelesen, verstanden und befolgt habe.

Weiters erkläre ich, dass die elektronische und gedruckte Version der eingereichten wissenschaftlichen Abschlussarbeit formal und inhaltlich identisch sind.

Datum 22.11.2022

Unterschrift Verfasser/in
Sabit Taygun Özdemir

Danksagung

Mein besonderer Dank gilt Herrn Univ.-Prof. Dipl.-Ing. Dr.techn. Martin Stockinger, dem Leiter des Lehrstuhls und meinem Betreuer Dipl.-Ing. Marcel Sorger, der mich immer tatkräftig unterstütz hat.

Des Weiteren möchte ich bei meiner Familie und meinen Freunden für die Unterstützung und den Rückhalt bedanken.

Kurzfassung

Mit fortschreitender Industrie 4.0. durchläuft der technische Bereich eine digitale Transformation, die durch das Hervorbringen neuer Technologien eine Vielzahl neuer Möglichkeiten bietet. Eine Enabler-Technologie der Industrie 4.0 ist das maschinelle Lernen (ML), welche sich als Teilbereich der künstlichen Intelligenz (KI) in den vergangenen Jahren zunehmend etablierte. Durch die vielfältige Anwendbarkeit zur Lösung von komplexen Aufgaben- und Problemstellungen wird ML zur Vorhersage und Berechnung von Werten, Erkennung von Zusammenhängen und Optimierungszwecken eingesetzt, was technische und wirtschaftliche Vorteile verspricht.

Zur Unterstützung der Entscheidungsfindung können durch den überwachten ML-Algorithmus Support Vector Machines (SVM) Daten bezüglich diverserer Charakteristika und Merkmalen klassifiziert werden.

Die Masterarbeit befasst sich mit der Lösung eines Klassifikationsproblems von Zeitreihendaten von Sensoren aus dem Bergbau durch die Entwicklung einer geeigneten SVM, um Daten mit Hilfe der SVM zuverlässig klassifizieren und daraus den entsprechenden Arbeitsschritten ableiten zu können.

In diesem Rahmen wurden statistische Datenanalysen, die Datenvorverarbeitung und die Bestimmung der Trainings- und Testdaten durchgeführt. Des Weiteren wurden die Leistungen verschiedener Kernelfunktionen zur Darstellung der ‘‘Cross Validation Accuracies‘‘ bei unterschiedlicher Kernelmethode evaluiert und die Optimierung der Ergebnisse durchgeführt.

Abstract

As Industry 4.0 progresses, the industry is undergoing a digital transformation that offers a multitude of new opportunities through the emergence of new technologies. One enabler technology of Industry 4.0 is machine learning (ML), which has become increasingly established as a subfield of artificial intelligence (AI) in recent years. Due to its diverse applicability for solving complex tasks and problems, ML is used for predicting and calculating values, recognizing correlations and optimization purposes, which promises technical and economic advantages.

To support decision-making, the supervised ML algorithm Support Vector Machines (SVM) can classify data with respect to more diverse characteristics and features.

This master thesis has the goal of solving a classification problem of time series data from sensors used in the mining industry by developing a suitable SVM to reliably classify data and derive from in the corresponding work steps using SVM.

In this framework, statistical data analysis, data pre-processing and determination of training and test data were performed. Furthermore, the performances of different kernel functions for the representation of ''Cross Validation Accuracies'' with different kernel method were evaluated and the optimization of the results was performed

Inhaltsverzeichnis

1. Einleitung 1
2. Stand der Technik 2
2.1. Klassifikationsmethoden 2
2.1.1. Support Vector Machine (SVM) 3
2.2. Verwandte Anwendungen in Bergbau 5
3. Grundlagen 6
3.1. Einführung in die Klassifizierung 6
3.2. Klassifikation 7
3.3. Generalisierung 8
3.3.1. Generalisierungsfähigkeit 8
3.3.2. Überprüfung der Generalisierungsfähigkeit 10
3.4. Theoretische und Mathematische Grundlagen der SVM 10
3.4.1. Mathematische Grundlagen der SVM 12
3.4.2. Lineare SVM für linear separierbare Daten 12
3.4.2.1 Formulierung des Optimierungsproblems 14
3.4.2.2 Lösung des Optimierungsproblems 15
3.4.2.3 Anwendung der Entscheidungsfunktion 17
3.4.3. Lineare SVM für nicht linear separierbare Daten 18
3.4.3.1 Anpassung des Optimierungsproblems 19
3.4.3.2 Anwendung der Entscheidungsfunktion 20
3.4.4. Kernel-SVM für nicht-linear separierbare Daten 20
3.4.5. Kernelmethoden 22
3.4.5.1 Linear-Kernel 22
3.4.5.2 Polynom-Kernel 23
3.4.5.3 Sigmoid-Kernel 23
3.4.5.4 RBF-Kernel 23
3.5. Zeitreihe 24
3.6. Software und Bibliotheken 26
3.6.1 Python 26
3.6.2 Verwendete Python-Bibliotheken 27
4. Datenanalyse und Evaluationskriterien 28
4.1. Datenerfassung (Rohdaten) 28
4.2. Erstellung von Teilmengen 29
4.3. Datenmodellierung 35

4.4. Standardisierung 36
4.5. Implementierung von SVM-Algorithmus 37
4.5.1 K-fold Cross Validation 39
4.5.2 GridSearchCV 41
4.6. Evaluationskriterien 45
5. Auswertung und Ergebnisse der Klassifizierung im Bergbau 49
5.1. Ergebnisse der Klassifizierung 53
6. Zusammenfassung und Ausblick 54
Abbildungsverzeichnis 55
Tabellenverzeichnis 56
Literaturverzeichnis 57
Anhang 62
Anhang A: Python Codes-Erstellung von Teilmengen 62
Anhang B: Python-Codes-Teilmengen 64
Teilmenge-Mittelwert 64
Teilmenge-Schiefe 68
Teilmenge-Wölbung 72

Abkürzungsverzeichnis

Symbol	Beschreibung
$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{C}$	Menge der komplexen Zahlen
$\mathbb{N}$	N Menge der natürlichen Zahlen
$\mathbb{R}^m$	Mehrdimensionale Räume
$\|X\|$	Anzahl der Elemente der endlichen Menge X
M^T	Transponierte Matrix A
$\|\|x\|\|$	Euklidische Norm, $\|\|x\|\| = \sqrt{x^T x}$
D	Menge der Daten $D = (X, Y) = T \cup E$
T	Menge der Trainingsdaten $T \subset D$
E	Menge der Testdaten $E \subset D,\ T \cap E = \{\}$
M	Anzahl der Trainingsdatenansätze $M = \|T\|$
m	Anzahl der Eingangsvariablen(Dimensionalität des Merkmalsraums) $m = dim(\chi)$
χ	Eingangsraum(Merkmalsraum) $\chi = \mathbb{R}^m$ mit Dimension $m \in \mathbb{N}$
X	Menge der Eingangsraumdaten (Rohdaten) $X \subset \chi$, $X = \{x_i, \dots, x_M\}$
x_i	i-ter Vektor der Trainingsdaten $x_i = \{x_1, \dots, x_M\},\ x \in \chi\ und\ 1 \leq i \leq M$
Ω_i	Klassen mit dem Index i
n	Anzahl der Klassen $n \in \mathbb{N}, n \geq 2$
Y	Ergebnisraum $Y = \mathbb{R}^n$
Y	Menge der Klassenlabels $Y = \{y_i, \dots, y_M\}$
y_i	Klassenlabels
S	Menge der Support Vector Indizes
U	U Menge der unbegrenzten Support Vector Indizes
H	Hyperebene
w	Normalvektor zur Hyperebene
$w^T x$	Skalarprodukt der Vektoren w und x
α_i, β_i	Lagrange-Multiplikatoren assoziiert mit x_i
α_i^*	Optimale Lagrange-Multiplikatoren
ξ_i	Schlupfvariable assoziiert mitx_i
C	Fehlergewicht (Größe des Margins)

b	Bias
$\mathcal{F}$	hochdimensionale SVM-Merkmalsraum, $dim(\mathcal{F}) > m$
$\phi(x): \chi \to \mathcal{F}$	Funktion zur Transformation von $x \in \chi$ in den Merkmalsraum $\mathcal{F}$
$K(x, z)$	Kernelfunktion äquivalent zu $\phi(x)^T \phi(z)$
γ	Parameter des RBF-Kernels
κ, θ	Parameter des Sigmoid-Kernels
ML	Machine Learning (dt. “Maschinelles Lernen“)
SVM	Support Vector Machine (dt. “ Stützvektormaschine“)
AI	Artificial Intelligence (dt, “ Künstliche Intelligenz“)
PCA	Principal Component Analysis
LDF	Linear Discriminant Functions

1. Einleitung

Heutzutage verwenden viele Unternehmen Techniken des Maschinellen Lernens (eng. “Machine Learning“, ML), um die Entscheidungsfindung zu unterstützen und Geschäftsprozesse zu automatisieren, indem sie aus den vorhandenen Daten lernen.

In der Welt, die jeden Tag mehr und mehr digitalisiert wird, gibt es viele ungeordnete Daten. Auch beim Bergbau ist es notwendig, komplexe Signaldaten nach Gemeinsamkeiten und Unterschieden zu klassifizieren. Auf diese Weise können Prognosen erstellt und weitere Ingenieure- und Produktion Prozesse unterstützt werden.

In diesem Zusammenhang hat sich am Lehrstuhl für Fertigungstechnik der Montanuniversität Leoben das Thema ergeben, wie aus realen physikalischen Systemen stammenden Zeitreihendaten für das Training, die Klassifikation und die Zukunftsvorhersage angewandt werden können.

Zu diesem Zweck wurde in dieser Arbeit eine Kombination aus traditioneller statistischer Datenanalyse mit Support-Vector-Machine (SVM), einem der hochmodernen Deep-Learning-Algorithmen, verwendet, dessen innere Arbeit leicht verständlich und keine Black-Box-Methode ist.

Kapitel 2 gibt einen Überblick über den Stand der Technik, Klassifikationsmethoden und die historische Entwicklung sowie Vor- und Nachteile von SVMs. Außerdem werden die Anwendungen von ML-Methoden im Bergbau erwähnt. In Kapitel 3 werden theoretische und mathematische Grundlagen von Klassifikation, Generalisierung, SVMs und Zeitreihendaten erläutert. In Kapitel 4 folgt der aktuelle Stand der Daten, die Aufbereitung der Daten und deren Umsetzung (Programmierung) mit Hilfe einer „Open Source“-Programmiersprache. Kapitel 5 enthält die Ergebnisse und deren Bewertung. In Kapitel 6 wird die Masterarbeit mit einer Zusammenfassung und einem Überblick über die Arbeit abgeschlossen.

2. Stand der Technik

Während der Industrie 4.0, der vierten industriellen Revolution, wird durch den zunehmenden Einsatz von Informationstechnologien die Digitalisierung und Vernetzung der Industrie durch den Einsatz neuer Technologien unterstützt. Derzeit findet die schrittweise Digitalisierung von Prozessen statt, wobei der Fokus vor allem auf der systemübergreifenden Kommunikation zwischen verschiedenen Prozessen sowie zwischen Personen und Prozessen liegt [1].

Der Rohstoff, der für die Nutzung dieser neuen Technologien benötigt wird, sind Daten. Daher ist die Datenerhebung und -verarbeitung in nahezu allen Industriebereichen zum Standard geworden. Ziel ist es, die aktuelle Situation besser zu verstehen, Prozesse zu optimieren, einzuordnen und Vorhersagen über die zukünftige Situation zu treffen.

Die Datenerfassung allein bietet jedoch nicht das Potenzial, den Herstellungsprozess zu verbessern, es sei denn, die Daten werden analysiert und aussagekräftige Informationen extrahiert. Dies kann mit einer Vielzahl von Datenanalysemethoden erfolgen.

Maschinelles Lernen wird im Bergbau, wie in allen anderen Bereichen, häufig eingesetzt, um den steigenden Anforderungen gerecht zu werden. Dementsprechend kommt der Klassifikations- und Vorhersage Entwicklung, die eine der Aufgaben des maschinellen Lernens ist, eine große Bedeutung zu.

2.1. Klassifikationsmethoden

Die Klassifizierung ist eine Aufgabe des maschinellen Lernens. Gängige Klassifikationsverfahren können in zwei große Kategorien eingeteilt werden [2]: überwachtes Lernen (supervised learning problem) und unüberwachtes Lernen (unsupervised learning problem). Bei dem überwachten Lernverfahren enthält der Algorithmus die Eingabevektoren der Trainingsdaten und ihre entsprechenden Zielvektoren, das heißt, die Zielvektoren sind die gewünschten Lösungen, die Labels (dt.“ Etiketten“) genannt werden. Es ist beabsichtigt, jeden Eingangsvektor einer von mehreren Kategorien zuzuordnen[3–5]. Besteht die gewünschte Ausgabe aus einer kontinuierlichen Variable, spricht man von Regression [4].

Einige wichtige überwachte Lernalgorithmen sind [5]:

- k-nächste Nachbarn
- Lineare Regression
- Logistische Regression
- Support Vector Machines (SVMs)
- Entscheidungsbäume und Random Forests
- Neuronale Netze (neuronale Netzwerkalgorithmen können unüberwacht oder halbüberwacht sein)

Wenn die Trainingsdaten aus einem Satz von Eingabevektoren ohne Zielwert bestehen, d. h. ohne Label hat, dann wird das unüberwachte Trainingsverfahren erwähnt [3–5]. Bei den unüberwachten Lernverfahren kann das Ziel darin bestehen, ähnliche Stichprobengruppen innerhalb des Datensatzes, die als Cluster bezeichnet werden zu entdecken, die Datenverteilung zu bestimmen oder um die Daten zur Visualisierung in einen hochdimensionalen Raum zu projizieren [4]. Einige wichtige unüberwachte Lernalgorithmen sind [5]:

- Affinity propagation
- K-Means
- DBSCAN (Density-Based Spatial Clustering of Applications with Noise)
- Hierarchische Clusteranalyse (HCA)
- Mean shift
- One-class SVM
- Isolation Forest
- BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies)

2.1.1. Support Vector Machine (SVM)

SVM (dt. „Stützvektormaschine“) sind überwachte Lernmodelle, die Daten mithilfe verschiedener Algorithmen zur Klassifizierung und Regression analysieren. Der SVM-Algorithmus wurde erstmals 1963 von Vladimir N. Vapnik und Alexey Ya Chervonenkis in den AT&T Bell Laboratories entwickelt und ist eines der besten Vorhersageverfahren, das auf der VC-Theorie (Vapnik- und Chervonenkis-Theorie) basiert [6]. In den folgenden Jahren wandten Bernhard Boser, Isabelle Guyon und Vladimir Vapnik die als „Kernel-Trick“ bekannte Methode auf einer Hyperebene an, um nichtlineare

Klassifikationsprobleme zu lösen. Im Jahr 1993 haben Corinna Cortes und Vapnik den „Soft Margin Classifier“ nahegelegt und im 1995 veröffentlicht [7],[6]. Die Vorteile und die Nachteile der SVM-Technik lassen sich wie folgt zusammenfassen [8]:

Vorteile der SVM;

- Durch die L2-Regularisierungsfunktion hat SVM eine gute Generalisierungsfähigkeit, dadurch wird eine Überanpassung vermeidet.
- SVM verarbeitet auch nichtlineare Daten mit den Kernel-Tricks.
- SVM kann sowohl bei Klassifikations- als auch Regressionsprobleme verwendet werden.
- Das SVM-Modell ist stabil. Kleine Änderungen in den Daten wirken sich nicht sehr auf die Hyperebene und damit auf die SVM aus.

Nachteile der SVM;

- Die Auswahl einer geeigneten Kernfunktion zur Verarbeitung nicht-linearer Daten ist keine leichte Aufgabe. Bei Verwendung eines hochdimensionalen Kernels können zu vielem Unterstützungsvektoren erzeugt werden, was die Trainingsgeschwindigkeit stark reduziert.
- SVM ist algorithmisch sehr komplex und der Speicherbedarf ist sehr hoch. Es wird viel Speicher benötigt, da alle Unterstützungsvektoren im Speicher gespeichert werden müssen, und diese Zahl steigt mit der Größe des Trainingssatzes.
- Die Skalierung des Merkmales von Variablen ist erforderlich, bevor SVM implementiert wird.
- SVM erfordert eine lange Trainingszeit für große Datasets und ist daher für sehr große Datasets nicht geeignet.
- Im Vergleich zum Decision Tree-modell ist das SVM-Modell schwer zu verstehen und zu interpretieren.

SVMs können verwendet werden, um Text- und Hypertextklassifizierung, Bildklassifizierung, Bildsegmentierung, Satellitendatenklassifizierung, Handschrifterkennung und viele weitere Klassifizierungsprobleme zu lösen.

Durch die Untersuchung von anormalem Verhalten in Daten können Ineffizienzen und Fehler im Prozess identifiziert werden, was zu einer effizienteren Nutzung von Energie und Ressourcen beiträgt und dadurch potenzielle Sicherheitsrisiken für Mitarbeiter und Kunden reduziert. Neben Datenerhebung, Datenverarbeitung und Datenanalyse Ziele wie Aufbau eines Kundennetzwerks,

Produkt- und Produktionsprozessänderungen, Produktinnovationen, Koordination und Standardisierung von Prozessen und Systemen, Automatisierung von Transportsystemen, Selbststeuerung und Automatisierung der Produktion kann mit der Philosophie der Industrie 4.0 realisiert werden [9].

2.2. Verwandte Anwendungen in Bergbau

ML-Modelle sind nützliche Werkzeuge, die zur Analyse von Rohdaten und zur Optimierung des Bergbaus eingesetzt werden können.

Durch die Analyse mikroseismischer Ereignisse in Bergbau können Gesteinsstabilität, Produktionsoptimierung, Materialeigenschaften (Erzidentifikation), Grenzlagen und viele ähnliche Aufgaben durchgeführt werden [10–12]. Diese mikroseismischen Ereignisse werden durch Gesteinsbruch, Bohren, Sprengen, Erzabbau usw. ausgelöst. Auch elektromagnetische Feldeffekte werden häufig in Bergwerken angetroffen [12]. Daher ist die Identifizierung und Klassifizierung mikroseismischer Ereignisse ein wichtiger Schritt in der Datenverarbeitung.

Viele automatische Klassifizierungsverfahren wie SVM [13], logistische neuronale Netze [14], Naive Bayes [15], künstliche neuronale Netze [16], Hauptkomponentenanalyse (PCA) [16], linear Diskriminante Funktion (LDF) [17] wurden vorgeschlagen, um mikroseismische Ereignisse zu unterscheiden und Vibrationen zu klassifizieren.

3. Grundlagen

Künstliche Intelligenz (KI) ist eine Integration von Informatik und Physiologie. In einfacher Sprache ist Intelligenz der rechnerische Teil der Fähigkeit, Ziele in der Welt zu erreichen. Intelligenz ist die Fähigkeit zu denken, sich etwas vorzustellen, sich etwas zu merken und zu verstehen, Muster zu erkennen, Entscheidungen zu treffen, sich an Veränderungen anzupassen und aus Erfahrungen zu lernen. Künstliche Intelligenz befasst sich damit, Computer dazu zu bringen, sich wie Menschen zu verhalten, und zwar auf menschlichere Art und Weise und in viel kürzerer Zeit, als ein Mensch dafür braucht [18].

In einfachen Worten bedeutet Lernen, entweder neues Wissen zu erwerben oder die Fähigkeiten des Einzelnen zu verbessern oder zu aktualisieren. Die Verbesserung der Fähigkeiten kann in biologischer Hinsicht als Verstärkung eines Musters neuronaler Verbindungen zur Ausführung der gewünschten Funktion interpretiert werden [19]. Maschinelles Lernen, ein Teilgebiet der KI, ist ein System, das statistische Methoden und Algorithmen verwendet, um eine Aufgabe auszuführen. ML erstellt mathematische Modelle, indem die Algorithmen auf die Trainingsdaten basieren, die nicht explizit vorprogrammiert sind [4]. Mit diesem Modell werden viele verschiedene ML-Aufgaben wie Clustering-, Klassifikations- und Vorhersagefunktionen durchgeführt.

3.1. Einführung in die Klassifizierung

Klassifikation kann als systematische Anordnung von Objekten in hierarchischen Systemen interpretiert werden, oft basierend auf formalen Kriterien, deren Objekte sich in verwandten Klassen unterscheiden [20,21]. Grundlage der Klassenzuordnung ist der Vergleich zwischen Klasseneigenschaften und Merkmalen, die Objekten zugewiesen oder berechnet werden. Die Zuordnung von Objekten in bestehende Klassen erfolgt in der Regel durch einen Klassifikator. Welches Klassifikationsmodell zu verwenden ist, hängt von den Daten und der jeweiligen Anwendung und den sich abzeichnenden Anforderungen an den Klassifikator ab. Die Zuordnung zu Klassen basiert auf Eigenschaften und der Fokus liegt auf der automatischen Klassifizierung durch ML. Das Ziel ist es, maschinelle Lernalgorithmen zu verwenden, die lernen, Beispielen in dem Problembereich ein Klassenlabel zuzuweisen [21–24]. Die automatische Klassifizierung mittels ML ist ein technischer Prozess, bei dem Objekte ohne menschliches Zutun anhand ihrer Merkmale klassifiziert werden. Die automatische Kategorisierung ist im Allgemeinen schneller und insgesamt genauer als die manuelle Kategorisierung. [21,25].

3.2. Klassifikation

Algorithmen, die mithilfe von Datenanalysen Klassifikationsregeln ableiten, werden als lernende Maschinen bezeichnet. Ein Algorithmus versucht aus den Eigenschaften der Eingabedaten $x_1, \dots, x_m$ und deren Ausgangsdaten y, funktionale Zusammenhänge abzuleiten. Die Datenmatrix X ist der assoziierte Klassenvektor mit den Klassenzuordnungen y, wobei $\mathrm{X} = (\mathrm{x}_1, \dots, \mathrm{x}_\mathrm{M})^\mathrm{T}$ und $\mathrm{y} = (\mathrm{y}_1, \dots, \mathrm{y}_\mathrm{M})^\mathrm{T}$ ist. Die Funktionen, die diese Beziehungen beschreiben, werden Hypothesen genannt. Der Ablauf des Lernens entspricht dem Finden der am besten geeigneten Hypothese, um die Klassifizierung von Daten zu beschreiben. Formal kann eine lernende Maschine als Funktion f angesehen werden, die Objektmerkmale aus dem Eingabedatensatz aus dem Eingaberaum X auf den Ergebnisraum Y abbildet [21,26]:

$$f: X \to Y \;\; f\ddot{u}r\; X \subset \mathbb{R}^m \quad und\; Y \subset \mathbb{R} \tag{3.1}$$

Der Verlauf der Lernphase, die Art der Trainings- und Testdaten sowie deren Einordnungsfähigkeit sind insbesondere von der Wahl des jeweiligen Algorithmus abhängig. Wie bereits erwähnt unterscheidet man grob zwischen unüberwachten und überwachten Klassifikationsverfahren nach der Lernspanne [4,9,21,25,27]:

Unüberwachte Methoden erfordern keine klassifizierten Trainingsdaten. Die Klassifizierung basiert nur auf Basis der Merkmalsdaten.

Es wird angenommen, dass die Trainingsmenge T eine Teilmenge von X ist. Sei X eine Menge nicht klassifizierter Datensatz, der aus dem Merkmalsraum $\mathcal{M}$ besteht. Weiterhin wird angenommen, dass der Merkmalsraum $\mathcal{M}$ sich aus den Merkmalsvektoren x_i zusammensetzt, wobei $x_i = (x_i, \dots, x_m)^T$ im Eingangsraum χ ein definierter Vektor mit *m* Merkmalen ist. Somit wird die Trainingsmenge T wie folgt definiert [7,21,26] ;

$$T = \{x_1, \dots, x_M\} \subset X \tag{3.2}$$

Im Vergleich zu einer einfachen Mengenanalyse versucht der Klassifikator, die Häufung von Vektorpunkten im mehrdimensionalen Merkmalsraum aus der Merkmalsverteilung zu erkennen und die Vektoren quantitativ zusammenzufassen. Die Anzahl der Mengen ergibt sich direkt aus den zu klassifizierenden Daten [21,28].

Überwachte Methoden erfordern Trainingsdaten, die der Klassifikator parametrisiert. Es wird davon ausgegangen, dass die Trainingsdaten die zu klassifizierenden Daten beschreiben. Dementsprechend sollten auch Trainingsdaten klassifiziert werden.

Es wird angenommen, dass die Trainingsmenge T eine Teilmenge der Objektmenge $\mathrm{D} = (\mathrm{X}, \mathrm{Y})$ ist, wobei X die Menge der Eingangsdaten und Y die Menge der Ausgangsdaten sind. Ebenfalls wird angenommen, dass (X, Y) eine Menge klassifizierter Daten, der aus Merkmalsraum $\mathcal{M}$ besteht. Merkmalsraum $\mathcal{M}$ setzt sich aus Paarungen $(\mathrm{x_i}, \mathrm{y_i})$ zusammen, wobei $x_i = (x_i, \dots, x_m)^T$ ein Vektor von m Merkmalen, die im Eingangsraum χ definiert und $\mathrm{y_i}$ ein Ausgangswert, der mit dem Vektor $\mathrm{x_i}$ zusammenhängt und repräsentiert eine Klasse aus dem Ausgangsraum Υ. Somit ist die Trainingsmenge T wie folgt definiert [21,26];

$$T = \{(x_1, y_1), \dots, (x_M, y_M)\} \subset (X, Y) \tag{3.3}$$

Überwachte Verfahren laufen in drei Phasen ab [23]:

1. Die überwachte Erfassung und Verarbeitung von Trainingsdaten.
2. Einen Klassifikator mit Trainingsdaten zu trainieren und mit Testdaten zu kontrolieren.
3. Tatsächliche Klassifizierung mit dem trainierten Klassifikator, d.h. Zuordnung eines beliebigen Objekts zu bestimmten Klassen.

3.3. Generalisierung

Beim ML wird durch das Lernen von Trainingsdaten eine Hypothese abgeleitet, sodass der Klassifikator unbekannte Testdaten mit möglichst geringem Fehler klassifizieren kann. Diese Verallgemeinerungsfähigkeit wird als Generalisierung bezeichnet. [21,29,30].

3.3.1. Generalisierungsfähigkeit

Wenn ein Klassifikator ein gutes Klassifikationsergebnis mit einer niedrigen Fehlerrate liefern kann, hat der Klassifikator eine gute Klassifikationsleistung. In einem solchen Fall wird die Generalisierungsfähigkeit als die Fähigkeit eines Klassifikators definiert, gute Trennergebnisse zu erzielen, indem unbekannte Daten an die Trainingsdaten angepasst werden.

Wenn die Trennungsfunktion nicht ausreichend zu den Trainingsdaten passt, geht die Klassifikationsleistung verloren. In diesem Fall ist eine sogenannte Über- oder Unteranpassung möglich.[4,7,21]:

Überanpassung(overfitting): Beim Training eines Klassifikators versucht man normalerweise, die Klassifizierungsleistung für die Trainingsdaten zu maximieren. Wenn der Klassifikator jedoch zu gut für die Trainingsdaten geeignet ist, wird die Klassifikationsfähigkeit für unbekannte Daten, d. h. die Generalisierungsfähigkeit, verschlechtert. Dieses Phänomen wird als Überanpassung bezeichnet, d.h. es gibt einen Kompromiss zwischen der Generalisierungsfähigkeit und der Anpassung an die Trainingsdaten. Overfitting behindert die gewünschte Generalisierungsleistung, also die Übertragbarkeit auf neue, ungelernte Daten (Abbildung 1).

Unteranpassung(underfitting): In Fällen, in denen der Klassifikator die Verteilung der Daten im Merkmalsraum nicht angemessen beschreiben kann, ist das beschreibende Modell zu einfach und kann die Struktur der Trainingsdaten nicht angemessen widerspiegeln.

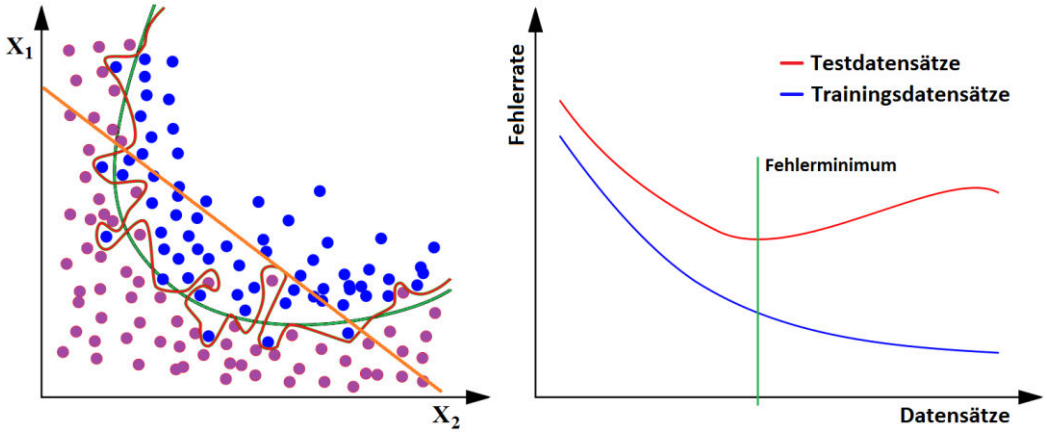


Abbildung 1: Überanpassung, Unteranpassung und Fehlerrate-Datensätze Diagramm [21,29]

Ein Beispiel für die Generalisierung im 2D-Merkmalsraum ist in Abbildung 1a angegeben. Die grüne Linie steht für eine gute Generalisierung, die rote Linie für Overfitting und die orange Linie für Underfitting. Das rechte Bild zeigt die Klassifikationsfehler in den Trainingsdaten (blaue Linie) und den Testdatensätzen (rote Linie). Die grüne vertikale Linie zeigt die optimale Anzahl von Trainingsdatensätzen mit minimalem Testdatenfehler an, d. h. die Anzahl von Datensätzen, die einen minimalen Fehler liefern können, wird durch Überprüfung mit Testdatensätzen für das Training

gesucht. Für das Training wird die Anzahl der Datensätze, die einen minimalen Fehler liefern können, durch Überprüfung mit Testdatensätzen gesucht. Dieser Zusammenhang ist in Abbildung 1b dargestellt.

3.3.2. Überprüfung der Generalisierungsfähigkeit

Es gibt einige Methoden, um den Generalisierungsgrad zu kontrollieren oder eine Überanpassung in einem frühen Stadium zu erkennen [21,23,29,31] . Einige davon sind die folgenden:

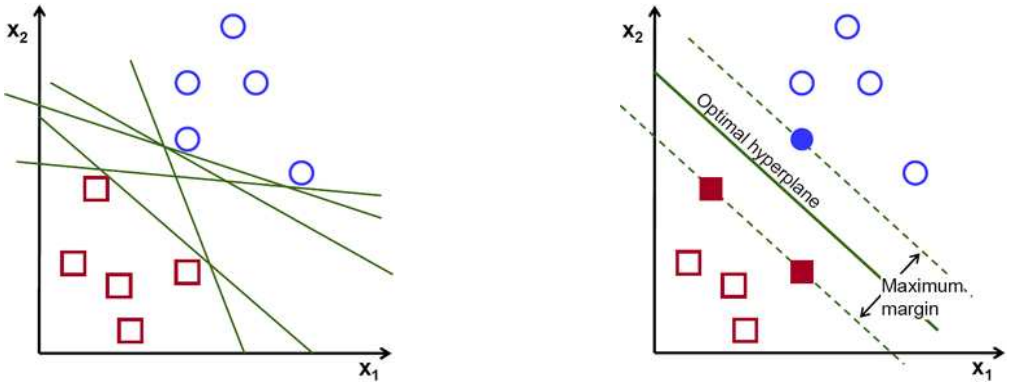
- Bei der Kreuzvalidierung werden die Trainingsdaten T in eine Anzahl K (in gleich große Teilmengen) aufgeteilt. Die i-te Teilmenge T_i wird als Testklasse verwendet und die verbleibenden Teilmengen werden zum Trainieren des Klassifikators verwendet. Nach dem Training wird eine Testklassifikation mit der Teilmenge T_i vorgenommen, dann werden die Ergebnisse mit den tatsächlichen Klassen verglichen, um eine Fehlerrate zu berechnen. Die Gesamtfehlerquote errechnet sich aus dem arithmetischen Mittel, also der Summe der Fehlerquoten in jedem Falte dividiert durch die Anzahl der Falte. Dieses Verfahren wird als *k-fache* Kreuzvalidierung (*engl. k-foldcross-validation*) bezeichnet.
- Eine Verallgemeinerung kann das Rauschen ignorieren. Wird den Trainingsdaten zunehmendes Rauschen hinzugefügt und gleichzeitig die Stabilität des Klassifikators überprüft, steht das Rauschen mit der Generalisierungsfähigkeit des Algorithmus in Zusammenhang und die Generalisierung kann bewertet werden.
- Zur unkomplizierten Validierung wird ein unabhängiger Tracking-Datensatz verwendet. Wenn die Klassifikationsleistung optimal ist, wird das Training beendet.

3.4. Theoretische und Mathematische Grundlagen der SVM

SVM zeigt die Beziehungen zwischen Eingabe- und Ausgabedaten. Dabei entwickelt SVM eine Funktion anhand einer bestimmten Menge an markierten Trainingsdaten. Diese Funktion wird *Mapping-Function* genannt. Eine Klassifizierung, Kategorisierung oder Regression kann mit einer Mapping-Funktion ausgedrückt werden. Nichtlineare Kernelfunktionen in SVM werden häufig verwendet, um die Eingabedaten in einen hochdimensionalen Merkmalsraum zu transformieren [32].

Bei der Klassifikation mit SVM wird zunächst ein Modell auf Basis des Trainingsdatensatzes berechnet. Im nächsten Schritt wird der noch fehlende Teil des Testdatensatzes anhand der Modell-

und Testdatenmerkmale geschätzt. Jedem Vektorelement im Vektorraum wird ein bestimmter Merkmalstyp zugewiesen, sodass jeder Vektor einer Beobachtung entspricht. Vektoren sind Listen von Daten, die einen Satz von Koordinaten im Raum darstellen. Der Merkmalsraum ist ein N-dimensionaler Vektorraum, wobei N gleich der Anzahl der Vektoren ist. Bei einer Klassifizierungsaufgabe, die eine Hyperebene D (oder H) beinhaltet, versucht der Algorithmus, die Klassen von Datenpunkten zu trennen. Die zugehörige Klasse von Datenpunkten muss natürlich vorher bekannt sein. Diese Ebene unterteilt das Gebiet in diese Klassen. Bei der Bestimmung des Ebenen wird darauf geachtet, dass beide Klassen den maximalen Abstand zu den verfügbaren Datenpunkten haben (Abbildung 2). Dieser maximale Abstand erhöht die Chance, dass ein neuer Datenpunkt auch richtig, also auf der rechten Seite der Ebene, klassifiziert wird. Die Datenpunkte, die der Ebene am nächsten liegen, werden Stützvektoren genannt. Nur durch diese Punkte wird die Lage der Hyperebene im Raum beeinflusst. Wenn ein neuer Datenpunkt näher an der Ebene liegt als die vorherigen Stützvektoren, ändert sich die Position der Ebene [6].



(a) (b)

Abbildung 2: Mögliche Hyperebenen (a) und optimal Hyperebene (b) [33]

Oft ist es nicht möglich, Datenpunkte linear aufzuteilen (Abbildung 3). Wie bereits erwähnt, wird in solchen Fällen ein Kernel-Trick verwendet. Damit wird der gesamte Vektorraum in eine höhere Dimension überführt und die Datenpunkte können nun linear geteilt werden [33,34].

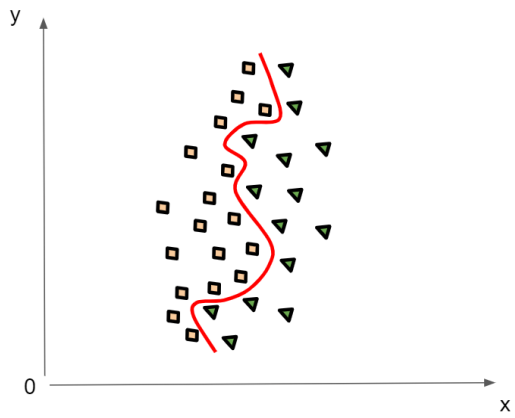


Abbildung 3: nicht-lineare Datentrennung [35]

3.4.1. Mathematische Grundlagen der SVM

In diesem Abschnitt werden grundlegende Ansätze, zuerst zweidimensionale lineare SVM, dann zweidimensionale nichtlineare SVM und mehrdimensionale SVM erläutert. Es erklärt auch die grundlegenden mathematischen Schritte, um die Entscheidungsfunktion für die Klassifizierung zu bestimmen.

Die Trainingsdaten (x_i, y_i), in der Menge D bestehen aus M Datenpaaren $(x_1, y_1), \dots, (x_i, y_M)$. $x_i \in \chi$, entspricht beobachteten Merkmalen und χ bezeichnet den Eingangsraum. Jeder Datensatz x_i hat m Eigenschaften im m-Dimensionalen Eingangsraum $\chi \subset \mathbb{R}^m$, interpretiert als Vektor $(x_1, \dots, x_m)^T$. Jeder dieser Vektoren bezieht sich auf einen Datenpunkt. *Y* gibt den Ergebnisraum an, wobei $y_i \in \{-1, +1\}$ den beobachteten Klassenlabels entspricht, sodass jedes Klassenlabel nur +1 oder -1 sein kann [21].

3.4.2. Lineare SVM für linear separierbare Daten

Bei der Berechnung der linearen Trennungsfunktion wird davon ausgegangen, dass die Daten linear und fehlerfrei getrennt werden können. In diesem Zusammenhang wird die Hard Margin Support Vector Machine erwähnt. In diesem Fall wird davon ausgegangen, dass die Punkten in den positiv markierten $D_+ = \{(x, y) \in D : y = +1\}$ und negativ markierten $D_- = \{(x, y) \in D : y = -1\}$Mengen exakt linear getrennt werden können und daher keine Klassifikationsfehler berücksichtigt werden sollten.

Die Definition einer Hyperebene lautet wie folgt [21,36]:

$$H(x) = w^T x + b = 0 \tag{3.4}$$

wobei ***w*** eine Gewichtsvektor und ***b*** Bias ist. Das Bias ***b*** ist eine Eigenschaft, die die Über- oder Unterschätzung einer Schätzfunktion vermisst, d.h. sie spiegelt den Abstand zwischen den vorhergesagten Daten und den tatsächlichen Daten als Ergebnis der Modellierung wider [37].

Wenn eine durch die Vektoren w und b definierte Hyperebene linear in die Klassen $\Omega_1 \, und \, \Omega_2$ unterteilt wird, dann gilt für alle Beispiele in den Trainingsdaten X_i für $i = 1, \dots, M$, [21,36]:

$$w^T x_i + b = \begin{cases} > 0 \;\; f\ddot{u}r \; alle \; x_i \; mit \; y_i = +1 \\ < \;\; 0 \;\; f\ddot{u}r \; alle \; x_i \; mit \; y_i = -1 \end{cases} \tag{3.5}$$

Da davon ausgegangen wird, dass die Trainingsdaten linear separiert werden können, braucht $w^T x + b = 0$ vorerst nicht berücksichtigt zu werden.

Das Ziel beim SVM-Training ist es, die H-Hyperebene optimal auf den Datenpunkt zu legen, so dass der Abstand zwischen der Hyperebene und den Punkten der Binärklassen$+1$ und -1 maximal ist. Auf beiden Seiten der optimalen Hyperebene, die zwischen den Stützvektoren der beiden Klassen liegt, sollte eine Grenze mit der größten Breite und frei von Vektoren aus den Trainingsdaten sein. Da die zu maximierende Kante der Grenzfläche auf beiden Seiten der Hyperebene genau Werten von ± 1 entsprechen muss, wird Formel (3.5) in folgende Ungleichungen umgerechnet [21,36].

$$w^T x_i + b = \begin{cases} \geq +1 \;\; f\ddot{u}r \; alle \; x_i \; mit \; y_i = +1 \\ \leq \; -1 \; f\ddot{u}r \; alle \; x_i \; mit \; y_i = -1 \end{cases} \tag{3.6}$$

Dies ist in Abbildung 4 dargestellt. Der maximierte Trennungsbereich erlaubt es dem SVM, diesem Bereich später unbekannte Vektoren korrekt zuzuordnen.

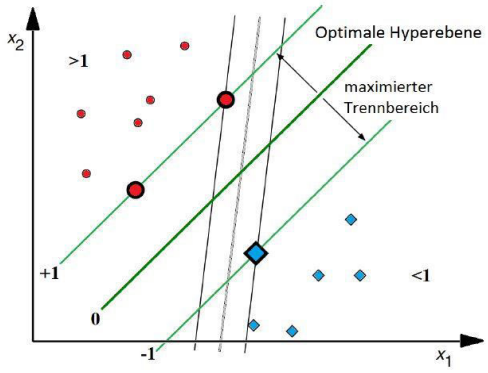


Abbildung 4: Optimal getrennte Hyperebene über drei Stützvektoren[36]

3.4.2.1 Formulierung des Optimierungsproblems

Der Hyperebenenabstand der Stützvektorpunkte am Rand des Trennfeldes ist durch die Gleichung (3.7) gegeben [21]

$$\left| \frac{W^T}{\|W\|} x_i + \frac{b}{\|W\|} \right| = \frac{1}{\|W\|} = \|W\|^{-1} \tag{3.7}$$

Die Breite des zu maximierenden Trennbereichs ist umgekehrt proportional zur Länge des Normalvektors $\left(\frac{2}{\|W\|}\right)$ der dem doppelten Abstand zwischen der Hyperebene und den Stützvektoren entspricht (Abbildung 4). Ein maximierter Trennflächenbedarf kann nun durch die Formulierung eines Optimierungsproblems mit Nebenbedingungen dargestellt werden [4,21,38] :

Optimierungsproblem;

$$(w^*, b^*) = \underset{w \in \mathcal{F}, b \in \mathbb{R}}{argmin} \frac{1}{2}\|W\|^2 \tag{3.8}$$

Mit der Bedingung;

$$y_i(w^T x_i + b) - 1 \geq 0 \quad f\ddot{u}r\ alle\ i = 1, \dots, M \tag{3.9}$$

Die geometrische Interpretation des Optimierungsproblems $\frac{1}{2}\|W\|^2 = \frac{1}{2}w^T w$ zeigt, dass sich die Hyperebene weiter von den Trainingsdaten entfernt, ohne sie voneinander zu trennen, wie in Abbildung 5a gezeigt. Daher müssen andere Einschränkungen die gesuchte Hyperebene zwischen

getrennten Klassen erzwingen. Dies wird durch die Forderung erreicht, dass alle orthogonalen Abstände auf beiden Seiten der Trennfläche zusammen maximiert werden [21,39]:

$$\sum_i d_i = \sum_{i=0}^{M} \left[y_i \left(\frac{w^T}{\|w\|} x_i + \frac{b}{\|w\|} \right) - \frac{1}{\|w\|} \right] \tag{3.10}$$

Abbildung 5b zeigt das Ergebnis dieser Einschränkungen. Die Hyperebene befindet sich nun zwischen den beiden Klassen auf der kleineren Probenseite. Schließlich bewegt sich die Hyperebene aufgrund des Zusammenwirkens der Minimierungs- und Maximierungsbedingungen zwischen den Punkten der beiden Klassen und bildet eine Trennzone (Abbildung 5c) [21,39].

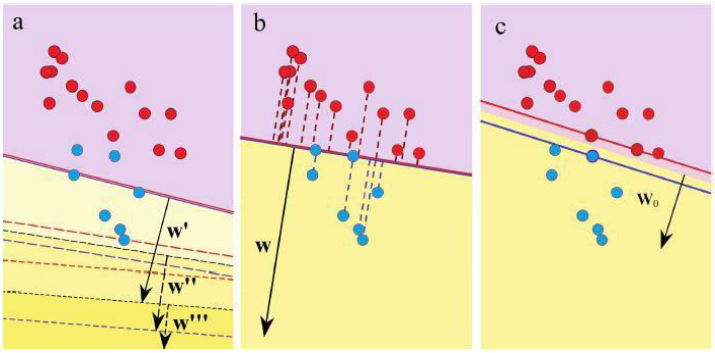


Abbildung 5: Drift der Hyperebene aufgrund der Optimierungsbedingungen [39]

Jede Bedingung ist nur dann eindeutig erfüllt, wenn y_i Klassenzuordnung aller Trainingsvektoren korrekt ist (sonst ist das Ergebnis $d_i \leq -1$) und keiner der Trainingsvektoren im Trennungsbereich liegt (sonst ist das Ergebnis $-1 < d_i < 0$) [21,39].

Mit Hilfe der obigen Herleitungen lassen sich optimale Bestimmungskriterien der Hyperebene $H: D\,(x) = w^T w + b = 0$ für die linear trennbaren Trainingsdaten der beiden Klassen bestimmen [21,25].

3.4.2.2 Lösung des Optimierungsproblems

Um das Optimierungsproblem zu vereinfachen, werden positive Lagrange-Multiplikatoren $\alpha = (\alpha_1, \ldots, \alpha_M)$ eingeführt und jede der Nebenbedingungen (3.9) multipliziert mit [21,40]

$$\alpha_i(y_i(w^T x_i + b) - 1) = 0 \tag{3.11}$$

Die Lagrange-Gleichung [4],

$$L(w, b, \alpha) = \frac{1}{2}\|W\|^2 - \sum_{i=0}^{M} \alpha_i(y_i(w^T x_i + b) - 1) \tag{3.12}$$

wird in Abhängigkeit vom $\sum_{i=0}^{M} \alpha_i y_i$ bezüglich w minimiert, bezüglich b minimiert oder maximiert und bezüglich α_i maximiert. Dafür setzt man die Ableitungen nach w und b jeweils gleich 0. Die Lösung der unter (3.12) formulierten Lagrange-Gleichung lautet [21,36];

Unter den Bedingungen

$$\frac{\partial L(w, b, \alpha)}{\partial w} = w = \sum_{i=0}^{M} \alpha_i y_i x_i = 0 \tag{3.13}$$

und

$$\frac{\partial L(w, b, \alpha)}{\partial b} = \sum_{i=0}^{M} \alpha_i y_i = 0 \tag{3.14}$$

und

$$\alpha_i\big(y_i(w^T x_i + b) - 1\big) = 0 \quad \text{für} \quad i = 1, \dots, M \tag{3.15}$$

und

$$\alpha_i \geq 0 \quad \text{für } i = 1, \dots, M \tag{3.16}$$

Karush-Kuhn-Tucker-Bedingungen [41] garantieren die Optimalität der Lösung und weisen jeder relevanten Nebenbedingung ein α_i zu, Nebenbedingungen, die wegen $\alpha_i = 0$ nicht zu scharf sind, werden nicht mehr in die Lösung aufgenommen [41] . Mit dem Einsetzen der Gleichungen (3.13) und (3.14) in die Lagrange-Gleichung (3.12) erhält man die Lösungsgleichung in dualer Form:

$$L(\alpha) = \sum_{i=0}^{M} \alpha_i - \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{3.17}$$

$$\sum_{i=0}^{M} \alpha_i y_i = 0 \tag{3.18}$$

und

$$\alpha_i \geq 0 \quad f\ddot{u}r\ i = 1, \dots, M \tag{3.19}$$

3.4.2.3 Anwendung der Entscheidungsfunktion

Die optimalen Lagrange-Multiplikatoren $\alpha_1^*, \dots, \alpha_M^*$ lassen sich aus den Gleichungen (3.17), (3.18) und (3.19) mit den Trainingsvektoren berechnen.

Da aus Gleichung (3.17) oder (3.12) folgt, dass die Vektoren aufgrund der Lagrange-Multiplikatoren α_i selektiv und nicht gleichermaßen an der Lösung teilnehmen, befinden sich die Daten, die positiv sind $\alpha_i > 0$, in der Menge der Support Vektor Indizes S. Alle anderen Vektoren $\alpha_i = 0$ haben keinen Einfluss auf das Modell [21,36].

In der Praxis wird nur $\alpha_i > \varepsilon$ verwendet, wobei ε ein kleiner positiver Wert ist. Als nächstes wird der optimale Gewichtsvektor $\mathrm{w}^*\ von\ S = \{i: \alpha_i > \varepsilon\},\ |S| \ll |X|$ gemäß Gleichung (3.13) bestimmt [21,38]:

$$w^* = \sum_{i=1}^{M} \alpha_i^* y_i x_i \tag{3.20}$$

Wenn die Formel (3.20) in (3.4) eingesetzt wird, wird die Entscheidungsfunktion [36];

$$D(x) = \sum_{i \in S} \alpha_i^* y_i\, x_i^T x + b \tag{3.21}$$

Der Bias b wird aus einem beliebigen Stützvektor x_i, durch die Gleichung $b = y_i - w^T x_i \, für \, i \in S$ oder aufgrund möglicher numerischer Ungenauigkeiten durch den Mittelwert aller Stützvektoren abgeleitet [21,36];

$$b^* = \frac{1}{|S|} \sum_{i \in S} (y_i - w^T x_i) \tag{3.22}$$

Die Klassifizierung für zwei Klassen $\Omega_1 \; and \; \Omega_2$ lautet [21];

$$Class(x) = \begin{cases} \Omega_1 & falls \; D(x) > 0 \\ \Omega_2 & falls \; D(x) < 0 \end{cases} \tag{3.22}$$

3.4.3. Lineare SVM für nicht linear separierbare Daten

Trainingsdaten können nicht vollständig linear getrennt werden, da die meisten realen Daten einigen Fehlern unterliegen und Fehlklassifizierungen durch eine strenge Klassifizierungsgrenze vermieden werden. Dazu kann eine einfache Modifikation von linear trennbaren Trainingsdaten mitunter mit einer Fehlklassifikation verbunden sein, obwohl Trainingsmuster bisher nicht linear trennbar sind. Diese SVM, bei denen die Klassifikationsgrenze aufgeweicht wird, werden "Soft-Margin-SVM" genannt. Hierzu werden positive Schlupfvariablenξ_i (engl. Slack variables) erwähnt. Diese Schlupfvariablen erfüllen die Funktion der Fehlerterme und repräsentieren die Abweichung des entsprechenden Vektors x_i von der Trennflächen (Abbildung 6) [21,36].

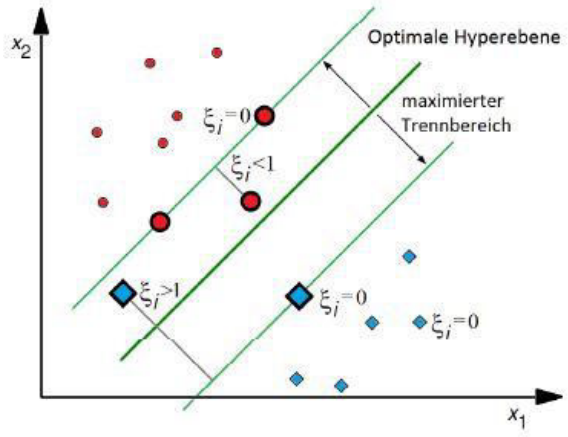


Abbildung 6: Anwendung einer Soft-Margin bei nicht-trennbaren Trainingsdaten [36]

Diese Verschiebungsvariablen mildern die strengen Beschränkungen (3.6), so dass die Trainingsvektoren auch in den maximalen Trennungsbereich enthalten sind. $\xi_i = 0$ gültig für korrekt klassifizierte Vektoren. $0 < \xi_i \leq 1$ gilt für Vektoren, die sich innerhalb des Trennbereichs und auf der korrekten Seite der Hyperebene befinden. $\xi_i > 1$gilt für Vektoren, die im Trennungsbereich liegen, aber bereits auf der falschen Seite der Hyperebene liegen. Sie gelten auch als falsch klassifiziert [4,21]. Die Abschätzung des Gesamtfehlers für alle Vektoren ist $\sum_i \xi_i$ und dementsprechend ist die Modifikation der Optimierungsproblem (Anpassung der Minimierungsbedingung) (3.7) wie folgt [21,36];

$$(w^*, b^*, \xi^*) = \underset{w \in \mathcal{F}, b \in \mathbb{R}^M}{argmin} \frac{1}{2}\|W\|^2 + \frac{C}{p}\sum_{i=1}^{M} \xi_i^p \tag{3.24}$$

Der Parameter $\boldsymbol{C}$ $(C > 0)$ regelt die Wirkung der Schlupfvariablen und wird somit zur Stellgröße für Soft-Margin-SVMs. $\boldsymbol{C}$ steht für Fehlergewicht und regelt sowohl die Größe des Trennbereichs als auch die Anzahl der Trainingsfehler. Der Parameter $\boldsymbol{p}$ nimmt man normalerweise 1 oder 2. Wenn $\boldsymbol{p}$ zunimmt, werden die Berechnungen komplexer. Derartige SVMs werden als „L1 Soft-Margin SVM“ bezeichnet [36]. p = 1 kommt am häufigsten zum Einsatz. Wenn p = 2, werden diese SVMs folgerichtig mit „L2 Soft-Margin SVM“ bezeichnet [21,36].

3.4.3.1 Anpassung des Optimierungsproblems

Die neuen Bedingungen nach den Änderungen in (3.24) lauten wie folgt [21,25];

$$\begin{aligned} w^T x_i + b &\geq +1 - \xi_i \ \text{für} \ \ y_i = +1 \\ w^T x_i + b &\leq -1 - \xi_i \ \text{für} \ \ y_i = -1 \end{aligned} \tag{3.25}$$

und somit

$$y_i(w^T x_i + b) \geq \ \ 1 - \xi_i \ \ \text{für} \ \ i = 1, \dots, M \tag{3.26}$$

$$\xi_i \geq 0 \qquad \text{für } i = 1, \dots, M \tag{3.27}$$

In diesem Fall lautet die neue Lagrange-Gleichung [21,25];

$$L(w,b,\phi,\alpha,\xi) = \frac{1}{2}\|w\|^2 + C\sum_{i=0}^{M}\xi_i - \sum_{i=0}^{M}\alpha_i(y_i(w^T x_i + b) - 1 + \xi_i) - \sum_{i=0}^{M}\beta_i\xi_i \qquad (3.28)$$

Die Lösung des Optimierungsproblems ist grundsätzlich die gleiche wie in den Gleichungen (3.13) bis (3.17). Die partiellen Ableitungen bezüglich w, b und ξ für $\xi = (\xi_1, \dots, \xi_M)$ sind wie folgt [21];

$$\frac{\partial L(w,b,\alpha,\xi)}{\partial \xi} = C - \alpha_i - \beta_i = 0 \qquad (3.29)$$

bewirken zusammen mit den KKT-Bedingungen eine Änderung der Bedingung (3.19) [21,38],

$$0 \le \alpha_i \le C \quad f\text{ür } alle\ i = 1, \dots, M \qquad (3.30)$$

3.4.3.2 Anwendung der Entscheidungsfunktion

Der optimale Gewichtsvektor w^* kann durch Gleichung (3.20) berechnet werden. Der b^*-Bias kann jedoch nicht mehr aus einem Stützvektor abgeleitet werden, da y_i-Werte einer Verteilung unterliegen können. Daher muss die optimale Abweichung b^*aus dem Mittelwert aller unbegrenzten Support-Vektor-Indizes $U = \{i: 0 < \alpha_i < C\}$berechnet werden [21,36].

$$b^* = \frac{1}{|U|}\sum_{i\in U}(y_i - w^T x_i) \qquad (3.31)$$

Für die Klassen $\Omega_1\ und\ \Omega_2$ sind s die Entscheidungsfunktion (3.21) und die Klassifizierungsregel (3.23) wie in der Hard-Margin Support Vektor Maschine.

3.4.4. Kernel-SVM für nicht-linear separierbare Daten

Wenn der Merkmalsvektor $x \in \mathbb{R}^m$ in einen höher dimensionalen Raum $\tilde{x} \in \mathbb{R}^l$mit $l > m$, durch einer nichtlinearen Abbildungsfunktion $\phi(x) = \big(\phi(x_1), \dots, \phi(x_M)\big)^T$ umgewandelt wird, bleiben die Gleichungen in (3.5), (3.6) und (3.25) weiterhin gültig, aber statt x wird mit $\phi(x)$ berechnet. Somit wird die Hypereben Gleichung $H(x)$ als nicht linearen Trennebene umformuliert [21,25].

$$H^{'}(x) = w^{T}\phi(x) + b \tag{3.32}$$

Eine lineare Trennung der transformierten Merkmalsvektoren im hochdimensionalen Raum ist anhand eines Beispiels in Abbildung 7 dargestellt.

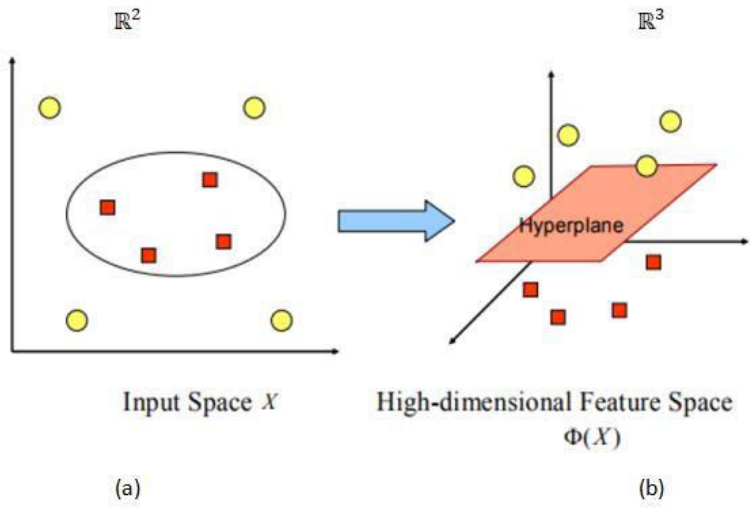


Abbildung 7: Lineare Trennung durch Transformation in einen höher dimensionalen Raum [42]

Mithilfe einer Abbildungsfunktion $\phi(x)$ werden nicht linear separierbare Daten (Abbildung 7a) in einen 3-dimensionalen Raum $\mathbb{R}^3$ transformiert (Abbildung 7b), in dem sie dann linear separierbar sind. Die Hyperebene im höher transformierten Raum entspricht einer nicht linearen Trennfunktion im Eingangsraum. In diesen Fall wird die Entscheidungsfunktion (3.21) in den hochdimensionalen Abbildungsraum $\mathbb{R}^l$ überführt [21].

$$D(x) = \sum_{i \in S} \alpha_i^* y_i \phi(x_i)^T \phi(x) + b \tag{3.33}$$

Die Berechnung des Skalarprodukts, das erforderlich ist, um große Datenmenge in den höchdimensionalen Raum umzuwandeln, ist komplex und außerdem es ist nicht einfach, eine Abbildungsfunktion $\phi(x)$ abzuschätzen, die eine lineare Trennbarkeit im hochdimensionalen Raum $\mathbb{R}^{\tilde{d}}$ ermöglicht. Stattdessen ist es viel einfacher, die Hyperebene aus dem Merkmalsraum zu rücktransformieren, indem die Trainingsdaten $x \in X$ in einen höchdimensionalen Merkmalsraum $\mathcal{F}$

transformiert und die Trennungshyperebene $H^{'}$ in diesem Merkmalsraum spezifiziert wird. Für diesen Ansatz werden die sogenannten Kernelmethoden oder Kernel-Tricks verwendet [21,38].

3.4.5. Kernelmethoden

Wenn die Skalarprodukte $\phi(x)^T\phi(z)$ von transformierten Vektoren in einem Algorithmus vorkommen, so lassen sie sich unter bestimmten Bedingungen auf Kernel-Auswertungen von nicht transformierten Vektoren induzieren. Dieses Verfahren kann angewendet werden, wenn eine Kernfunktion K wie folgt definiert [21,43]:

$$\mathrm{K}(x,z) = \phi(z)^T\phi(x) \tag{3.34}$$

Die Kernfunktion $\mathrm{K}(x,z)$ ist gleich dem Skalarprodukt der Abbildungsfunktion $\phi(z)^T\phi(x)$, aber ϕ in K muss nicht explizit berechnet werden. Die Abbildungsfunktion ϕ konvertiert die Eingangsvektoren $(x,z) \in X\ und\ X \subset \mathbb{R}\ oder\ \mathbb{C}$ in den höchdimensionalen Merkmalsraum $\mathcal{F}$ [21,43].

Wie bereits erwähnt, besteht einer der Vorteile von SVM darin, dass sie verwendet werden, um die Generalisierungsleistung zu verbessern und die Klassifizierungsgeschwindigkeit zu erhöhen, sodass die Kernel-Auswahl und/oder -Entwicklung eine wichtige Rolle bei SVM spielt. In dieser Studie wurde kein neuer Kernel entwickelt, es wurde jedoch weiterhin mit den vier am häufigsten verwendeten Kerneltypen gearbeitet.

3.4.5.1 Linear-Kernel

Der lineare Kernel ist der einfachste Kernel, er hat keine Flexibilität und trennt die Daten als Linie.
Es wird verwendet, wenn das Klassifizierungsproblem linear trennbar ist. In diesem Fall sollte eine Konvertierung der Daten in einen höherdimensionalen Merkmalsraum vermieden werden. Wenn die Parametrierung ungültig ist, fördert dies nur eine Überanpassung der SVM. Es ist wie folgt definiert. [21,36]:

$$K(x,z) = x^T z \tag{3.35}$$

3.4.5.2 Polynom-Kernel

Ein Polynomkern wird häufig bei einfachen nichtlineare Klassifikationsprobleme verwendet. Dieser Kernel funktioniert wie ein linearer Kernel, indem er c auf Null ($c = 0$) und d auf Eins ($d = 1$) setzt. Der Polynomkern entspricht den Bedingungen von Mercer. Ein Nachteil von Polynomkernen ist, dass die Optimierung der Parameter c und d den Trainingsprozess verlangsamt. Wenn der Grad d der Funktion übermäßig erhöht wird, neigt der Klassifikator dazu, etwas überanzupassen. Der Kernel ist definiert als [21,36]:

$$K(x, z) = (x^T z + c)^d \tag{3.36}$$

3.4.5.3 Sigmoid-Kernel

Sigmoid-Kernel werden in Bereichen verwenden, in denen Klassifikator basierend auf neuronalen Netzwerken verwendet werden. Tatsächlich sind SVMs, die Sigmoidkerne verwenden, einem zweischichtigen Perzeptron äquivalent. Ein Nachteil bei Sigmoidkernen ist, dass der Optimierungsalgorithmus in bestimmten Situationen nicht konvergiert. Dies liegt daran, dass Sigmoidfunktionen nur unter bestimmten $\kappa\ und\ \theta$ Parametern Mercer-Bedingungen gehorchen, da Sigmoidkerne nur als bedingt positiv definiert sind [21,25,44]. Der Sigmoid-Kernel ist wie folgt definiert:

$$K(x, z) = \tanh(\kappa x^T z + \theta) \tag{3.37}$$

3.4.5.4 RBF-Kernel

Eine RBF-Kernel (Radial Basic Function) ist eine Funktion, die den Abstand von Punkt x zum Ursprung $\phi(x) = \|x\|$ oder zu einem beliebigen Punkt z $\phi(x) = \|x - z\|$ darstellt. RBF-Kernel können wie folgt definiert werden; [21]

$$K(x, z) = f(\, d(x, z)\,) \tag{3.38}$$

wobei d eine Abstandsfunktion auf X und f eine Funktion in R_0^+ ist. Eine Abstandsfunktion wird üblicherweise aus der Norm $d(x, z) = \|x - z\|$ abgeleitet [40]. Aufgrund der relativ einfachen

Parametrisierung der SVM sind RBF-Kernel die erste Wahl, wenn nichtlineare Trainingsdaten zur Verfügung stehen. RBF-Kernel bzw. Gauss-Kernel, zum Beispiel

$$K(x, z) = \exp(-\gamma \|x - z\|^2) \tag{3.39}$$

für $\gamma > 0$. Oft wird der Gauss-Kern $\gamma = \frac{1}{2\sigma^2}$ $für$ $\sigma \neq 0$ parametrisiert [21,40].

3.5. Zeitreihe

Eine Zeitreihe (engl. Time Series) ist ein sequenzieller Satz von Datenpunkten, die typischerweise über aufeinanderfolgende Zeiten gemessen werden. Sie ist mathematisch definiert als ein Satz von Vektoren x(t), t = 0,1,2,... wobei t die verstrichene Zeit darstellt. Die Variable x(t) wird als Zufallsvariable behandelt. Die während eines Ereignisses in einer Zeitreihe aufgenommenen Messungen sind in einer geeigneten chronologischen Reihenfolge angeordnet [45,46].

Eine Zeitreihe, die Aufzeichnungen einer einzelnen Variablen enthält, wird als univariant bezeichnet. Wenn jedoch Aufzeichnungen von mehr als einer Variablen berücksichtigt werden, wird dies als multivariat bezeichnet. Eine Zeitreihe kann kontinuierlich oder diskret sein. Eine stetige Zeitreihe besteht aus Messungen, die zu jedem Zeitpunkt erhalten wurden. Im Gegensatz dazu enthält eine unstetige Zeitreihe Messungen zu unterschiedlichen Zeitpunkten. Als kontinuierliche Zeitreihe können beispielsweise Temperaturmesswerte, Strömung eines Flusses, Konzentration eines chemischen Prozesses etc. aufgezeichnet werden. Andererseits können die Bevölkerung einer bestimmten Stadt, die Produktion eines Unternehmens, die Wechselkurse zwischen zwei verschiedenen Währungen diskrete Zeitreihen darstellen. Üblicherweise werden in einer diskreten Zeitreihe die aufeinanderfolgenden Beobachtungen in gleichmäßig beabstandeten Zeitintervallen wie stündlichen, täglichen, wöchentlichen, monatlichen oder jährlichen Zeitabständen aufgezeichnet. Allgemein wird angenommen, dass eine Zeitreihe von vier Hauptkomponenten beeinflusst wird, die sich von den beobachteten Daten unterscheiden lassen. Diese Komponenten sind: Trend, Zyklizität, Saisonalität und Unregelmäßigkeit [45,47].

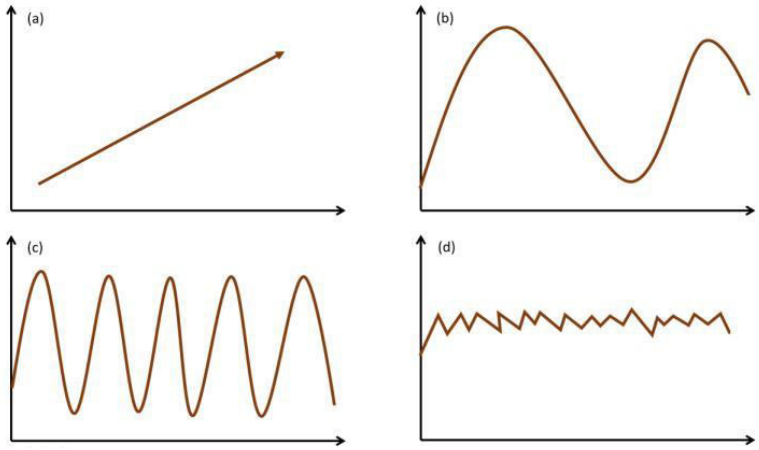


Abbildung 8: Zeitreihenkomponenten: (a)Trend; (b) Saisonal; (c) Zyklisch; (d) Unregelmäßig [47]

Der allgemeine Trend einer Zeitreihe, über einen längeren Zeitraum zuzunehmen, zu fallen oder zu stagnieren, wird als Trend bezeichnet. Zum Beispiel Bevölkerungswachstum, Anzahl der Häuser in einer Stadt usw. Während die verwandten Reihen einen steigenden Trend zeigen, sind Sterblichkeitsraten, Epidemien usw. Bei den verwandten Serien ist ein Abwärtstrend zu beobachten. Saisonale Schwankungen in einer Zeitreihe sind Schwankungen innerhalb der Jahreszeiten. Zum Beispiel steigen die Eisverkäufe im Sommer und die Verkäufe von Wollstoffen im Winter. Der saisonale Wechsel ist ein wichtiger Faktor, um die richtigen Pläne für die Zukunft zu machen. Zyklische Variation in einer Zeitreihe beschreibt mittelfristige Veränderungen in der Reihe, die durch sich zyklisch wiederholende Bedingungen verursacht werden. Die meisten Wirtschafts- und Finanzzeitreihen weisen eine Form zyklischer Veränderungen auf. Unregelmäßige oder zufällige Änderungen in einer Zeitreihe resultieren aus unvorhersehbaren Effekten, die nicht regelmäßig sind und sich auch nicht in einer bestimmten Reihenfolge wiederholen. Diese Veränderungen werden durch Krieg, Streik, Erdbeben, Überschwemmung, Revolution und ähnliche Ereignisse verursacht [48].

Es gibt kein definiertes statistisches Verfahren zur Messung zufälliger Schwankungen in einer Zeitreihe. Aufgrund der Auswirkungen dieser vier Komponenten werden üblicherweise zwei unterschiedliche Modelle für eine Zeitreihe verwendet. Multiplikative und additive Modelle [48].

Multiplikatives Modell $$Y(t) = T(t) \times S(t) \times C(t) \times I(t) \quad (3.40)$$

Additives Modell $$Y(t) = T(t) + S(t) + C(t) + I(t) \quad (3.41)$$

Hier ist Y(t) die Beobachtung und T(t), S(t), C(t) und I(t) sind jeweils der Trend, die saisonale, zyklische und unregelmäßige Variation zum Zeitpunkt t.
Das multiplikative Modell basiert auf der Annahme, dass die vier Komponenten einer Zeitreihe nicht notwendigerweise unabhängig sind und sich gegenseitig beeinflussen können, während beim additiven Modell davon ausgegangen wird, dass die vier Komponenten voneinander unabhängig sind [48].

In der Praxis wird ein geeignetes Modell an eine gegebene Zeitreihe angepasst und die entsprechenden Parameter werden anhand der bekannten Datenwerte geschätzt. Das Verfahren der Anpassung einer Zeitreihe an ein geeignetes Modell wird als Zeitreihenanalyse bezeichnet [15]. Sie umfasst Methoden, mit denen versucht wird, die Natur der Zeitreihe zu verstehen, und ist häufig für Zukunftsprognosen und Simulationen nützlich. Bei der Zeitreihenanalyse werden Beobachtungen aus der Vergangenheit gesammelt und analysiert, um ein geeignetes mathematisches Modell zu entwickeln, das den zugrunde liegenden Datenerzeugungsprozess für die Reihe erfasst [49,50]. Das Modell wird dann verwendet, um zukünftige Ereignisse zu klassifizieren und/oder vorherzusagen.

Dieser Ansatz ist besonders nützlich, wenn nur wenig Wissen über das statistische Muster der aufeinanderfolgenden Beobachtungen vorhanden ist oder wenn es kein zufrieden stellendes Erklärungsmodell gibt.

3.6. Software und Bibliotheken

3.6.1 Python

Python ist eine interpretierte, interaktive, objektorientierte Programmiersprache. Die einfache, auf Einrückungen basierende Syntax macht die Sprache leicht zu erlernen und zu merken. Dies verleiht ihm die Eigenschaft, eine Sprache zu sein, in der mit der Programmierung begonnen werden kann, ohne Zeit mit den Details der Syntax zu verschwenden. Seine modulare Struktur unterstützt Klassenzeichenfolgen (System) und beliebige Datenfeldeingaben. Es läuft auf fast jeder Plattform (Unix, Linux, Mac, Windows, Amiga, Symbian). Software kann in vielen Bereichen wie Systemprogrammierung, Benutzeroberflächenprogrammierung, Netzwerkprogrammierung, Anwendungs- und Datenbanksoftwareprogrammierung mit Python entwickelt werden [51].
Die Entwicklung begann 1990 von Guido Van Rossum in Amsterdam. Entgegen der landläufigen Meinung erhielt es seinen Namen nicht von einer Schlange, sondern von der Show Monty Python's Flying Circus einer sechsköpfigen britischen Comedy-Gruppe namens Monty Python, die Guido Van

Rossum sehr liebte. Heute wird es durch die Bemühungen von Freiwilligen gepflegt, die sich um die Python Software Foundation versammelt haben [52].

3.6.2 Verwendete Python-Bibliotheken

In der Umsetzung dieser Arbeit werden einige Python Bibliotheken verwendet. Diese sind;

- **Matplotlib:** Matplotlib ist eine Bibliothek mit einer einfachen Schnittstelle zum Erstellen hochwertiger Grafiken.[53].
- **Numpy:** Numpy ist eine Python-Bibliothek zur Durchführung arithmetischer Operationen an Arrays (einschließlich mehrdimensionaler) [54].
- **Scikit-learn:** Scikit-learn ist eine Bibliothek, die in vielen unterschiedenen Bereichen von ML für verschiedene Aufgaben wie Datenvorverarbeitung und Datenanalyse verwendet wird [55].
- **Pandas:** Pandas ist eine für Python geschriebene Bibliothek, die Datenstrukturen und Operationen speziell für die Bearbeitung numerischer Tabellen und Zeitreihen für Aufgaben wie Datenbearbeitung und -analyse bereitstellt.[56].
- **Seaborn:** Seaborn ist eine auf Matplotlib basierende Bibliothek, die eine grafische Schnittstelle zum Anpassen der generierten Diagramme bietet, die besonders gut geeignet sind zum Erstellen mathematischer Diagramme.[57]

4. Datenanalyse und Evaluationskriterien

In diesem Kapitel werden die verfügbaren Daten dargestellt und die grundlegenden Vorprozesse erläutert. Als nächstes wird der SVM-Algorithmus mit verschiedenen Kernel-Funktionen angewendet und schließlich wird das erfolgreichste und zuverlässigste Ergebnis mit GridSearch und k-fold Cross Validierung evaluiert.

4.1. Datenerfassung (Rohdaten)

Die in dieser Arbeit verwendeten Daten gehören zu einer Echtzeit-Zeitreihe. Die Daten wurden zu unterschiedlichen Zeitpunkten nicht aufeinanderfolgend gemessen und anschließend als „Bohren“, „Sprengen“ und „Schuttern“ vorklassifiziert und zur Verfügung gestellt. Abbildung 9 zeigt die Messwerte des Datensatzes als Zeitreihen für drei verschiedene Klassen.

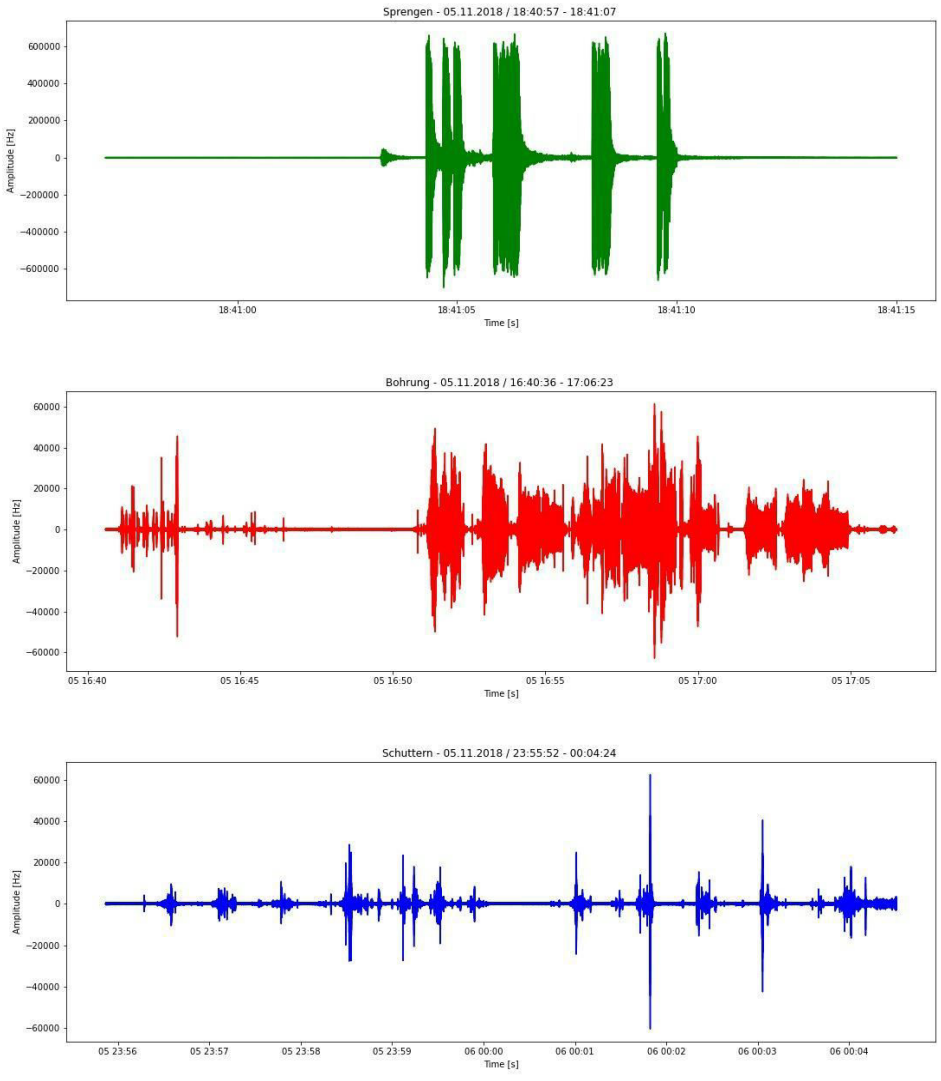


Abbildung 9: Zeitreihen für Sprengen (oben), Bohrung (mittig) und Schuttern (unten)

Jeder Datensatz enthält eine Spalte mit dem Zeitpunkt und 12 Spalten mit dem Messwert. Der Messwert kommt jeweils von einem Geophon, welches Schwingungen im Untergrund misst und wird mit einem sequenziell ansteigenden Echtzeit-Zeitstempel (alle 500 Mikrosekunden) aufgezeichnet. Dadurch entsteht ein für jedes Ereignis;

- Bohren (B); Vorbereitung der Sprengladung,
- Sprengung (S); Sprengung des Gesteins,
- Schuttern (Sch); Räumung des gesprengten Gesteins,

gültiger Dataframe von 13 Spalten (Abbildung 10). Der Hauptzweck besteht darin, zukünftige Ereignisse anhand dieser 12 Signalwerte zu klassifizieren und/oder vorherzusagen.

	Time	1	2	3	4	5	6	7	8	9	10	11	12	Events
0	2018-11-22 00:28:36.000000	-7.05513	3164.080	-272.296	147.2460	20.31790	-4190.850	175.7310	116.3650	3.977000	-6.85375	167.3960	-2.421000	B
1	2018-11-22 00:28:36.000500	-9.37565	3146.210	-308.647	154.3410	22.12870	-2666.080	179.8550	133.1590	3.473240	-7.44844	162.4300	0.925858	B
2	2018-11-22 00:28:36.001000	-6.33487	2915.740	-334.003	154.5430	22.22680	-981.323	176.5190	145.1250	8.120570	-9.22567	130.9670	0.170018	B
3	2018-11-22 00:28:36.001500	-19.99370	2650.590	-352.184	169.5290	19.69600	-114.716	168.6220	156.1490	-3.328000	-9.70778	96.0741	-4.788020	B
4	2018-11-22 00:28:36.002000	-7.50479	2360.750	-325.624	158.9440	18.79410	850.051	164.0740	157.3870	6.627580	-4.04160	73.5627	0.129706	B
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8853995	2018-11-22 01:42:22.997500	-8.48611	1845.010	-228.511	180.5510	7.07350	2454.170	94.0150	97.3233	6.247000	9.29348	49.0657	-1.515070	B
8853996	2018-11-22 01:42:22.998000	-2.51096	1483.560	-196.348	167.3760	9.17937	3508.510	87.7064	91.6146	-4.848980	3.67176	54.8903	-4.584650	B
8853997	2018-11-22 01:42:22.998500	5.36434	1171.080	-176.172	139.0560	14.71450	4027.390	80.2539	80.9533	0.636238	-2.00643	45.9562	-6.667180	B
8853998	2018-11-22 01:42:22.999000	2.50343	811.536	-165.393	111.1450	13.54510	4344.480	51.6062	73.1924	2.819410	-1.99240	13.8955	-7.556470	B
8853999	2018-11-22 01:42:22.999500	-1.34796	530.023	-162.519	96.0538	4.03644	3528.320	29.0884	67.7240	-4.158690	3.29270	-37.2887	-10.014400	B

8854000 rows × 14 columns

Abbildung 10: „Bohrung“-Werte gemessen zwischen 00:28:36-01:42:22 Uhr am 22.11.2018

4.2. Erstellung von Teilmengen

Gesamtgröße des Datenpakets beträgt ca. 200 GB (ca. 100 Millionen Zeilen). Das trainieren mit einer solchen Datengröße würde übermäßig viele Rechenressourcen in an Anspruch nehmen, weshalb entschieden wurde nur die Teilmengen der Daten zum Training der SVM heranzuziehen. Dazu wurden Mittelwert (mean), Wölbung (kurtosis) und Schiefe (skewness)-Werte jeder Sensormessung für jedes Zeitintervall berechnet, d.h. die Daten wurden in drei Teilmengen erhoben (Anhang A).

Der Zweck der Verwendung dieser drei Parameter besteht darin, die Verteilung und Variabilität der Daten zu charakterisieren und Ausreißer zu erkennen.

Der Mittelwert ist eigentlich ein Modell des Datensatzes. Der Mittelwert möglichst verringert den Fehler bei der Prognose eines Wertes im Datensatz, d. h. er erzeugt die niedrigste Fehlerrate aller Werte. Es ist auch ein Maß, bei dem die Summe der Abweichungen vom Mittelwert immer Null ist [58].

Schiefe ist der Mangel an Symmetrie. Es gibt drei Arten von Schiefe: Positiv, Negativ und Symmetrisch. Die Symmetrie eines Datensatzes wird durch das Aussehen der Verteilung links und rechts der Mitte ausgedrückt. Sieht die Verteilung gleich aus, ist sie symmetrisch. In diesem Fall ist die Schiefe der Verteilung gleich 0. (Abbildung 11) [59].

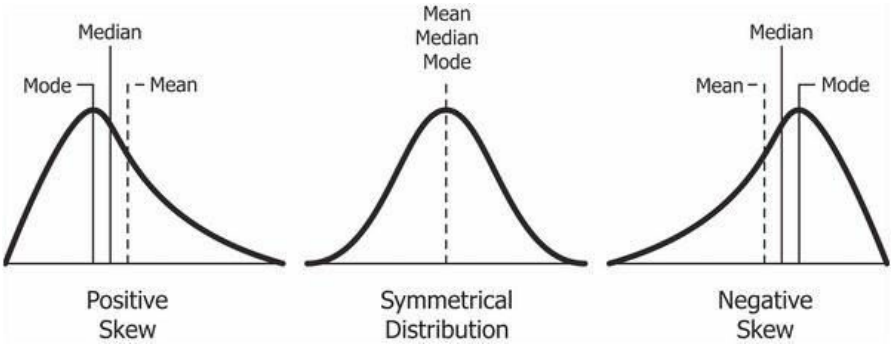


Abbildung 11: Arten von Schiefen [60]

Bei positiver Schiefe, bei der Mittelwert und Median größer als der Modus sind, ist der Schwanz auf der rechten Seite der Verteilung länger oder dicker, während bei negativer Schiefe Mittelwert und Median kleiner als der Modus und grafisch der Schwanz auf der Verteilung sind links der Verteilung ist länger oder dicker als der Schwanz rechts.[60].

Wenn die Schiefe zwischen -0,5 und 0,5 liegt, sind die Daten ziemlich symmetrisch. Wenn die Schiefe zwischen -1 und -0,5 (negativ verzerrt) oder zwischen 0,5 und 1 (positiv verzerrt) liegt, sind die Daten mäßig verzerrt. Wenn die Schiefe kleiner als -1 (negativ schief) oder größer als 1 (positiv schief) ist, sind die Daten stark verzerrt [60].

Wölbung oder auch Kurtosis gennant, ist ein Maß dafür, ob die Daten relativ zu einer Normalverteilung schwer- oder leichtschwänzig sind. Das heißt, Datensätze mit hoher Wölbung neigen dazu, schwere Schwänze oder Ausreißer zu haben. Datensätze mit geringer Wölbung weisen leichte Schwänze oder keine Ausreißer auf [59].

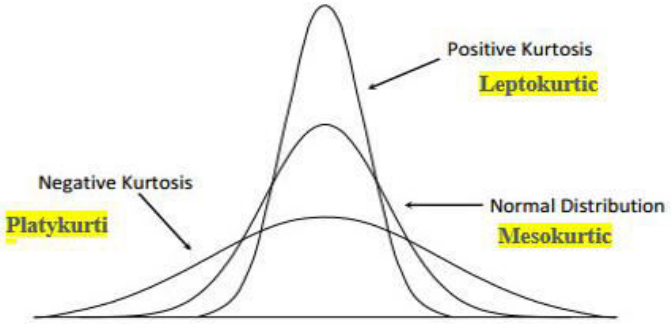


Abbildung 12: Art von Kurtosis [61]

Die überschüssige Kurtosis wird in der Statistik und Wahrscheinlichkeitstheorie verwendet, um den Kurtosiskoeffizienten mit dieser Normalverteilung zu vergleichen. Überschüssige Kurtosis kann positiv (leptokurtische Verteilung), negativ (platykurtische Verteilung) oder nahe Null (mesokurtische Verteilung) sein (Abbildung 12). Da Normalverteilungen eine Kurtosis von 3 haben, wird die überschüssige Kurtosis berechnet, indem die Kurtosis um 3 subtrahiert wird [61].

Leptokurtik (Wölbung > 3) hat sehr lange und dünne Schwänze, was bedeutet, dass die Wahrscheinlichkeit von Ausreißern größer ist. Positive Kurtosis-Werte zeigen an, dass die Verteilung einen Höhepunkt aufweist und dicke Ausläufer aufweist. Eine extrem positive Kurtosis weist auf eine Verteilung hin, bei der sich mehr Zahlen in den Schwänzen der Verteilung als um den Mittelwert befinden.

Platykurtik (Wölbung < 3) mit einem niedrigeren Schweif und gestreckt um Mittelschwänze bedeutet, dass die meisten Datenpunkte in großer Nähe zum Mittelwert vorhanden sind. Eine platykurtische Verteilung ist im Vergleich zur Normalverteilung flacher (mit weniger Spitzen).

Mesokurtik (Kurtose = 3) ist die gleiche wie die Normalverteilung, was bedeutet, dass die Kurtosis nahe 0 ist. In Mesokurtic sind die Verteilungen mäßig breit und die Kurven haben eine mittlere Spitzenhöhe [61].

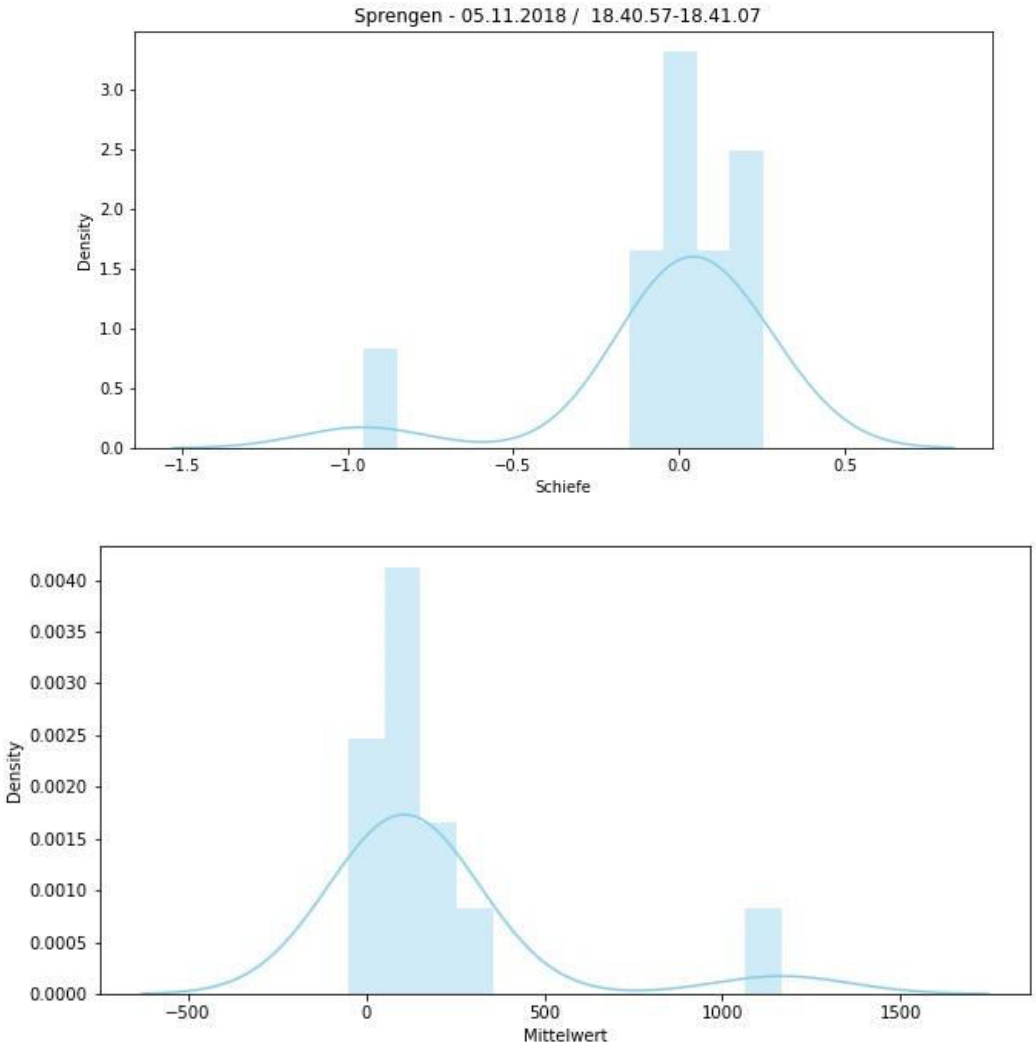
Die Daten, die jeweils von 12 Sensoren in alle 500 Mikrosekunden gemessen wurden, wurden als die oben genannten Parameter in drei Teilmengen mit 12 verschiedenen Werten in Form von insgesamt 156 Datenzeilen transformiert. Diese Operationen wurden für alle drei Klassen (Bohren, Schuttern, Sprengen) durchgeführt (Abbildung 13, für andere Teilmengen siehe Anhang B)

	1	2	3	4	5	6	7	8	9	10	11	12	Events
0	-0.002	-1.513	-0.344	0.492	-0.754	-1.061	-0.719	-0.359	-0.647	-1.777	-0.851	0.311	B
1	-0.632	-2.250	-0.749	-0.260	-0.331	-0.966	-0.881	-0.649	-0.448	-0.908	-0.929	-1.238	B
2	-0.740	-1.941	2.185	-0.207	-0.187	-1.495	-0.812	-0.700	-0.749	-0.300	-0.269	-1.453	B
3	-0.547	-1.377	1.439	-0.154	-0.511	-1.193	-0.613	-1.361	-0.557	-1.166	-0.197	-0.238	B
4	-0.835	0.298	-0.681	-0.679	-0.351	-0.163	-0.491	-0.717	-0.918	-0.121	-0.750	-0.858	B
...	...	...	...	...	...	...	...	...	...	...	...	...	...
151	-0.444	0.259	-0.321	-1.361	0.603	-1.034	-0.301	-1.185	0.066	0.153	-0.605	-0.280	Sch
152	-0.492	-1.362	0.564	-1.861	0.254	-0.802	-0.872	-1.466	-0.170	-0.136	-0.582	-0.312	Sch
153	-0.506	-0.925	1.539	-1.571	-0.053	-1.243	-0.511	-0.764	-0.171	-0.119	-0.879	-0.395	Sch
154	-0.398	0.722	-0.281	-0.272	0.363	1.114	-0.510	-0.702	-0.088	-0.120	-0.677	-0.394	Sch
155	-0.419	-0.345	0.880	-1.580	0.463	-2.324	-0.419	-0.751	-0.106	-0.184	-0.889	-0.410	Sch

156 rows × 13 columns

Abbildung 13: Teilmenge Mittelwert

Als Beispiel zeigen Abbildung 14-16 die Schiefe-, Mittelwerts- und Wölbung der Sprengen-, Bohren- und Schutterndaten, die in bestimmten Zeitintervallen am 5. November 2018 gemessenen wurden.



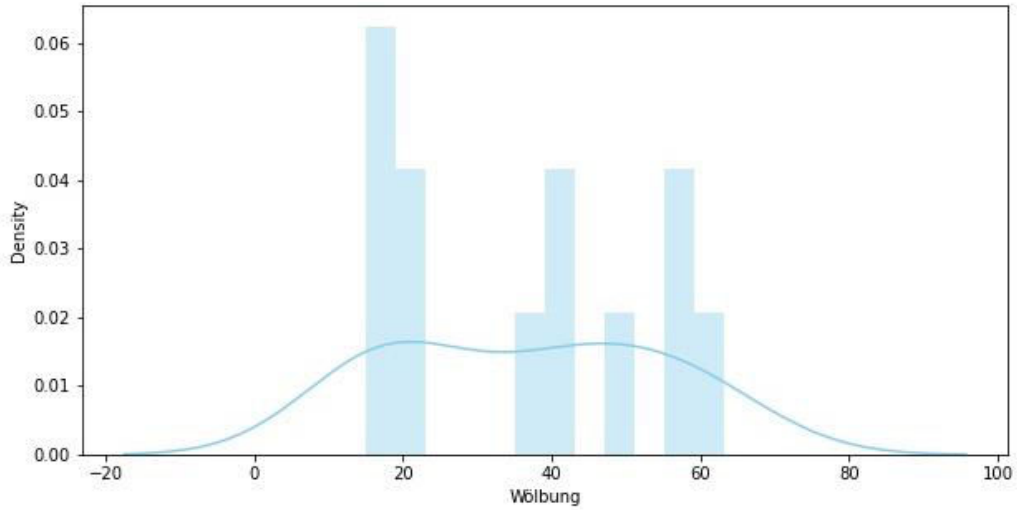
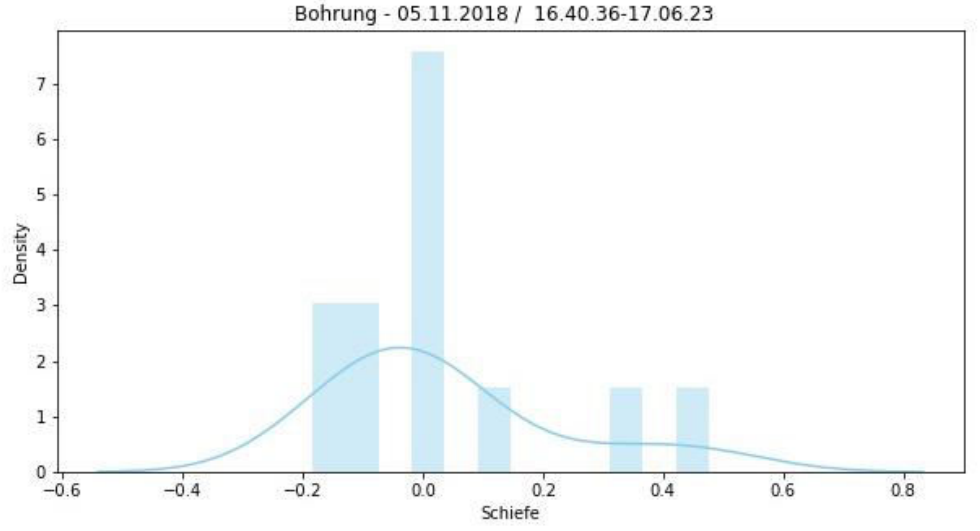
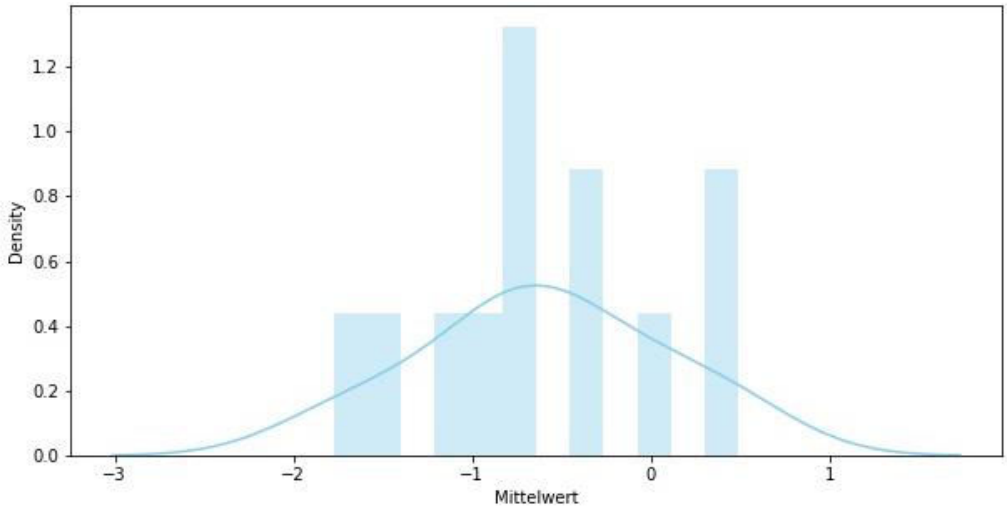


Abbildung 14: Darstellung von Signalen der Sprengen-Klasse für die Teilmengen Schiefe-, Mittelwert- und Wölbung.





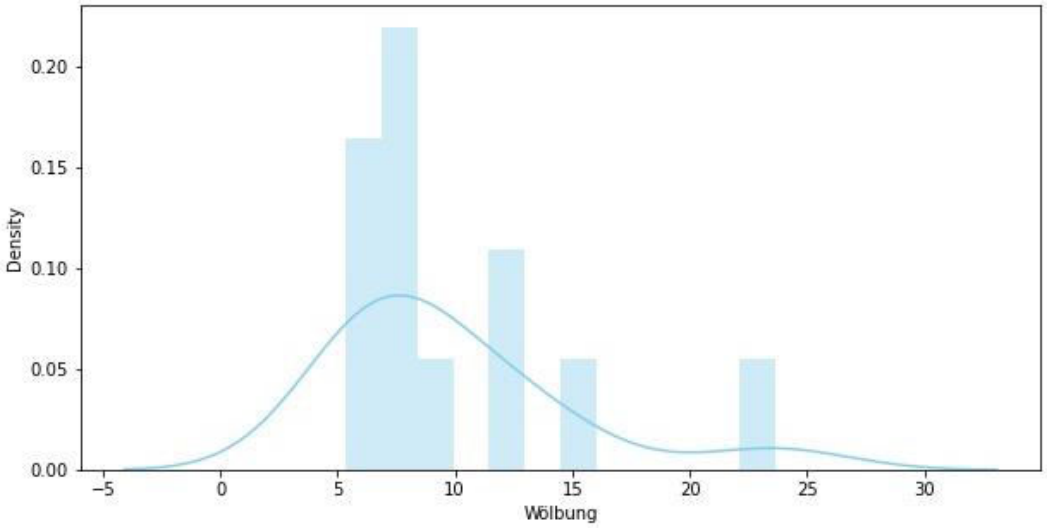
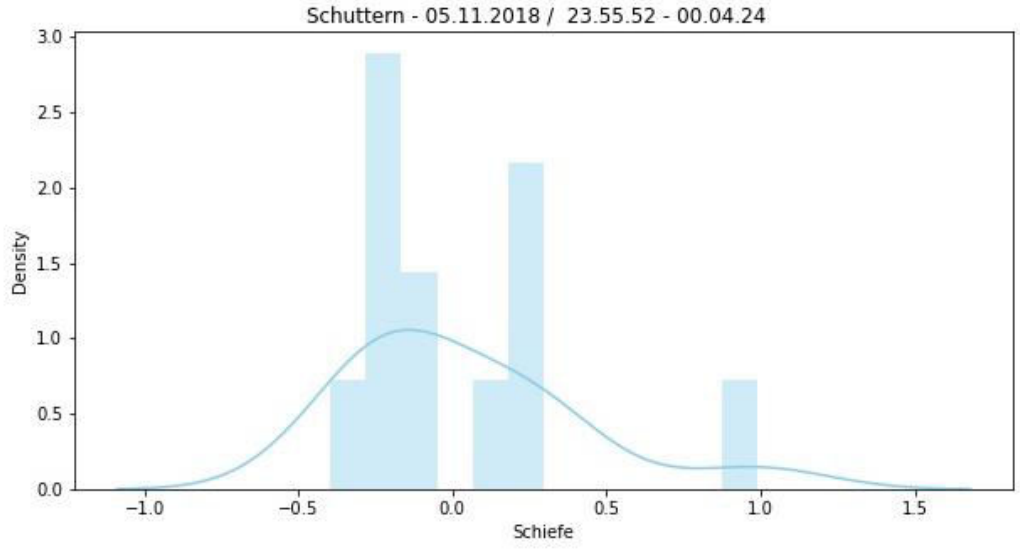
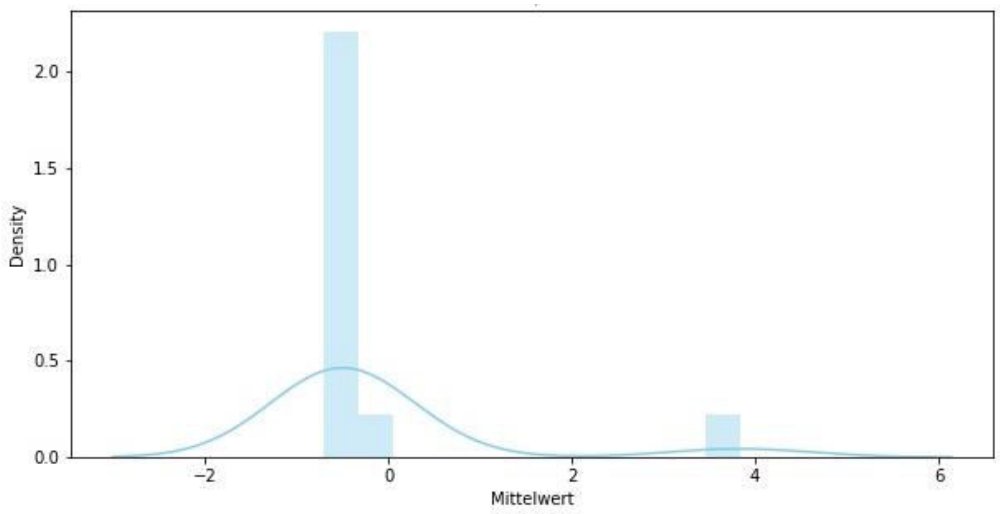


Abbildung 15: Darstellung von Signalen der Bohren-Klasse für die Teilmengen Schiefe-, Mittelwert- und Wölbung.





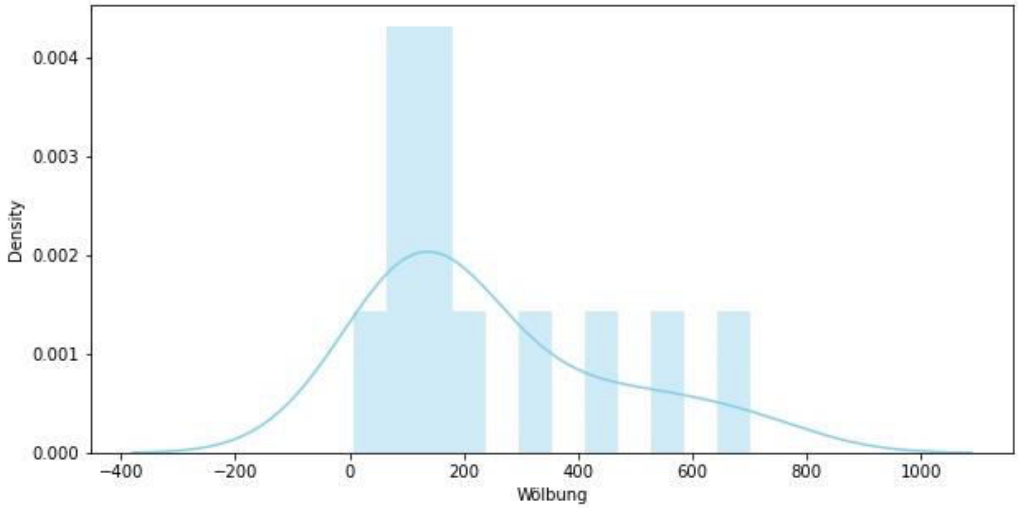


Abbildung 16: Darstellung von Signalen der Schuttern-Klasse für die Teilmengen Schiefe-, Mittelwert- und Wölbung.

Diese Werte in den Teilmengen spielen eine entscheidende Rolle bei der Bestimmung der Klassen, da sie für jede Klasse unterschiedliche Werte haben werden. Von dem Model(Algorithmus) wird erwartet, dass er als Prozentsatz zeigt, wie erfolgreich jede dieser Teilmengen beim Identifizieren und/oder Vorhersagen der Klassen sein wird.

4.3. Datenmodellierung

Wie vorher erwähnt, besteht der Hauptzweck von ML darin, Algorithmen zu erstellen, die aus vorhandenen Daten lernen, dadurch hat man die Möglichkeit zukünftige Ereignisse vorherzusagen. Die Arbeitsprinzipien dieser Algorithmen basieren auf der Erstellung eines mathematischen Modells aus den Eingabedaten. Dieses erstellte mathematische Modell wird verwendet, um Vorhersage- oder Entscheidungsfindungsfunktionen durchzuführen. Die Eingabedaten, die zum Erstellen des Modells verwendet werden, sind in der Regel in mehrere Datensätze unterteilt [4,62]. Bei der Erstellung des Modells werden drei Datensätze verwendet: Trainings-, Validierungs- und Testsätze. Der Trainingsdatensatz ist ein Datensatz, der aus Beispielen besteht, die im Lernprozess verwendet werden, und wird verwendet, um beispielsweise die Parameter eines Klassifikators anzupassen [63,64]. Beim Aufbau eines guten Vorhersagemodells verwendet der Lernalgorithmus den Trainingsdatensatz die am besten geeigneten Kombinationen von Variablen zu bestimmen und zu lernen. Der aus diesem Datensatz gewonnene überwachte Lernalgorithmus wird für die Klassifizierung verwendet. [65]. Ziel ist es, ein fundiertes Modell zu erstellen, das neue und unbekannte Daten gut generalisieren kann [66]. Anschließend wird die Genauigkeit des Modells bei

der Klassifizierung neuer Daten an „neuen“ Proben durch Validierung und Testdatensätze evaluiert [67]. Um das Risiko von Problemen wie Überanpassung zu verringern, sollten Proben aus Validierungs- und Testdatensätzen nicht zum Trainieren des Modells verwendet werden [67].

Der Testdatensatz ist ein Datensatz, der unabhängig vom Trainingsdatensatz ist, aber derselben Wahrscheinlichkeitsverteilung wie der Trainingsdatensatz folgt. Ist ein Modell passend zum Trainingsdatensatz und auch zum Testdatensatz, spricht man von einer minimalen Überanpassung. Eine bessere Anpassung des Trainingsdatensatzes im Vergleich zum Testdatensatz weist normalerweise auf eine Überanpassung hin. Daher ist ein Testsatz einfach ein Satz von Stichproben, die verwendet werden, um die Leistung eines genau spezifizierten Klassifikators zu bewerten [63,64]. Dazu wird das endgültige Modell verwendet, um die Klassifizierungen der Proben im Testsatz vorherzusagen. Diese Schätzungen werden mit den tatsächlichen Klassifizierungen der Proben verglichen, um die Genauigkeit des Modells zu bewerten [65]. Wenn Validierungs und-Testdatensätze zusammen genutzt werden, wird der Testdatensatz im Allgemeinen bevorzugt, um das Modell während des Validierungsprozesses zu evaluieren. Wenn der Originaldatensatz in Trainings- und Testdatensätze aufgeteilt wird, kann der Testdatensatz das Modell nur einmal auswerten. [68]. Bei dem Kreuzvalidierungsverfahren wird jedoch die Summe der Trainings- und Testgenauigkeit, die von jeder Falten erhalten wird, gemittelt, um eine Über- und Unteranpassung minimal zu halten, so dass eine effektive Genauigkeit erhalten werden kann. Ein Validierungsdatensatz ist ein Datensatz von Beispielen, die zum Festlegen der Hyperparameter eines Klassifikators verwendet werden. Um eine Überanpassung zu vermeiden, ist es notwendig, zusätzlich zu den Trainings- und Testdatensätzen einen Validierungsdatensatz zu haben, wenn irgendein Klassifikationsparameter angepasst werden muss [66,67]. In dieser Arbeit werden die Daten in 80 % Trainingssatz und die restlichen 20 % Testsatz aufgeteilt, sodass die SVM als Klassifikator fungiert, um vorherzusagen, ob ein Ereignis eintreten wird.

4.4. Standardisierung

Die Skalierung (engl. Scaling) eines Datensatzes ist eine häufige Anforderung für ML. Die Standardisierung kommt ins Spiel, wenn die Eigenschaften des Eingabedatensatzes große Unterschiede zwischen ihren Bereichen aufweisen oder einfach in verschiedenen Maßeinheiten (z. B. Pfund, Meter, Meilen usw.) gemessen werden. Typischerweise erfolgt dies durch Entfernen des Mittelwerts und Skalieren auf Einheitsvarianz. Ausreißer können jedoch häufig den Mittelwert oder die Varianz der Stichprobe negativ beeinflussen. Der Hauptvorteil der Skalierung besteht darin, Features in größeren numerischen Bereichen zu vermeiden, die kleinere numerische Bereiche

dominieren. Ein weiterer Vorteil besteht darin, numerische Schwierigkeiten bei der Berechnung zu vermeiden. Große Merkmalswerte können numerische Probleme verursachen, da Kernel-Werte oft von inneren Produkten von Merkmalsvektoren abhängen, zum Beispiel Linear-Kernel und Polynomial-Kernel. Es wird empfohlen, jedes Merkmal linear auf den Bereich [−1, +1] oder [0, 1] zu skalieren. Natürlich ist es notwendig, die gleiche Methode zu verwenden, um sowohl Trainings- als auch Testdaten zu skalieren. SVMs bestimmen die Position der Trennebene, indem es während des Trainings den Abstand der Ebene zu den entsprechenden Stützpunkten maximiert (siehe Kapitel 3.5). Um eine Dominanz einzelner Merkmale zu vermeiden und alle Merkmale mit gleichem Gewicht in Distanzberechnungen einzubeziehen, sollten die Trainingsdaten so skaliert werden, dass alle Merkmalswerte auf einen ähnlichen Wertebereich abgebildet werden. Dazu steht eine Reihe von Standardisierungsmöglichkeiten zur Verfügung, die die Merkmale auf gleiche Skalen- und Größenordnungen bringen und damit die Merkmale gleichzeitig klassenübergreifend vergleichbar machen (Anhang B) [69–73].

4.5. Implementierung von SVM-Algorithmus

Nach Aufteilung der Daten in Teilmengen und Durchführung der notwendigen Datenvorverarbeitung können nun die Klassifikationsleistungen mit SVM unter Verwendung verschiedener Kernelfunktionen verglichen werden.

Zunächst wurde die Klassifizierung jeder Teilmenge manuell durchgeführt, dann wurden die optimalen Parameter erreicht (GridsearchCV, siehe 4.4.1). Anschließend wurde mit diesen Daten die beste Test- und Vorhersagegenauigkeit erzielt. Mit Hilfe des „Klassifizierungsberichts“ (Classification_Reports) wurden jedoch andere Metrikern untersucht.

Die Ergebnisse von vier verschiedenen Kernel-Funktionen für die Teilmengen Mittelwert, Wölbung und Schiefe sind in Tabelle1-3 dargestellt. Unter Verwendung eines Linearen Kernels erreichte das Modell eine Vorhersagegenauigkeit von 68 % für die „Mittelwert –Teilmenge“. Die Genauigkeit der RBF, Polynomial und Sigmoid Kernel beträgt 43%, 40% und 34 % (Anhang B)(Tabelle 1).

Kernel	Train accuracy	Test accuracy
Linear	0.82	0.68
RBF	0.56	0.43
Polynomial	0.66	0.40
Sigmoid	0.45	0.34

Tabelle 1: Ergebnisse von Kernelfunktionen für Mittelwert

Die Kernel-Ergebnisse in den Schiefe und Wölbung Teilmengen sind wie folgt;

Kernel	Train accuracy	Test accuracy
Linear	0.70	0.62
RBF	0.87	0.65
Polynomial	0.98	0.75
Sigmoid	0.34	0.28

Tabelle 2: Ergebnisse von Kernelfunktionen für Schiefe

Kernel	Train accuracy	Test accuracy
Linear	0.97	0.90
RBF	0.98	0.90
Polynomial	0.98	0.84
Sigmoid	0.58	0.75

Tabelle 3: Ergebnisse von Kernelfunktionen für Wölbung

Wie aus den Tabellen 1-3 ersichtlich ist, ergaben der Linear-Kern und der RBF-Kernel die höchsten Testgenauigkeiten mit 0,90 in der Wölbung-Teilmenge. Allerdings ist keiner dieser Werte aktuell verlässlich. Wie den Tabellen zu entnehmen ist, liegen manche Testwerte über den Trainingswerten und die Differenz zwischen Trainings- und Testergebnis ist groß.

Dies weist auf das Problem des Underfitting und Overfitting hin. Um das Modell zu entwickeln und den am besten geeigneten Parameter auszuwählen, sind eine Reihe zusätzlicher Prozesse erforderlich.

4.5.1 K-fold Cross Validation

Kernelauswahl und andere Parameter können weder aus den Daten noch durch geschickte Wahl eines bestimmten Kerneltyps vorhergesagt werden, da diese Parameter keine intuitive Bedeutung haben [74]. Stattdessen wird empfohlen, die k-fache Kreuzvalidierung zu verwenden, die aus allen Ergebnissen die maximale „Cross-Validation Accuracy“ auswählt [1]. Diese Methode ermöglicht die Auswahl des Modells mit dem geringsten Überanpassungsgrad.
Die K-fach-Kreuzvalidierung ist definiert als eine Methode zur Schätzung der Leistung eines Modells auf unsichtbaren (Test Daten) Daten. Es wird empfohlen, diese Technik zu verwenden, wenn die Daten knapp sind und eine gute Schätzung des Trainings- und Verallgemeinerungsfehlers erforderlich ist, um die Aspekte wie Underfitting und Overfitting zu verstehen. Diese Technik wird für die Hyperparameter-Abstimmung verwendet, damit das Modell mit dem am besten geeigneten Hyperparameterwert trainiert werden kann, was in den folgenden Abschnitten erläutert wird. Es handelt sich um eine ersatzlose Resampling-Technik. Der Vorteil dieses Ansatzes besteht darin, dass jedes Beispiel genau einmal für Training und Validierung (als Teil eines Testfaltens) verwendet wird. Wie bereits erwähnt, wird diese Technik verwendet, da sie dazu beiträgt, eine Überanpassung zu vermeiden, die auftreten kann, wenn ein Modell mit allen Daten trainiert wird. Durch die Verwendung der k-fachen Kreuzvalidierung sind wir in der Lage, das Modell auf k verschiedenen Datensätzen zu "testen", was dazu beiträgt, dass das Modell verallgemeinerbar ist.

In dieser Technik wird Folgendes zum Trainieren, Validieren und Testen des Modells ausgeführt [75]:

1. Das Dataset ist in Trainings- und Testdataset aufgeteilt.
2. Der Trainingsdatensatz wird in K-Faltungen aufgeteilt.
3. Aus den K-Falten wird (K-1) Falte für das Training verwendet
4. Eine Falte wird für die Validierung verwendet
5. Das Modell mit spezifischen Hyperparametern wird mit Trainingsdaten (K-1-Falten) und Validierungsdaten als 1-fach trainiert. Die Leistung des Modells wird aufgezeichnet.
6. Die obigen Schritte (Schritt 3, Schritt 4 und Schritt 5) werden wiederholt, bis jeder K-Falz für Validierungszwecke verwendet wurde. Aus diesem Grund wird es als k-fache Kreuzvalidierung bezeichnet.

7. Schließlich werden der Mittelwert und die Standardabweichung der Modellleistung berechnet, indem alle in Schritt 5 berechneten Modellwerte für jedes der K-Modelle verwendet werden.
8. Schritt 3 bis Schritt 7 wird für verschiedene Werte von Hyperparametern wiederholt.
9. Schließlich werden die Hyperparameter, die zum optimalsten Mittelwert führen, und der Standardwert der Modellbewertungen ausgewählt.
10. Das Modell wird dann mit dem Trainingsdatensatz trainiert (Schritt 2) und die Modellleistung wird auf dem Testdatensatz berechnet (Schritt 1).

Das Bild unten zeigt die Schritte 2 bis 7;

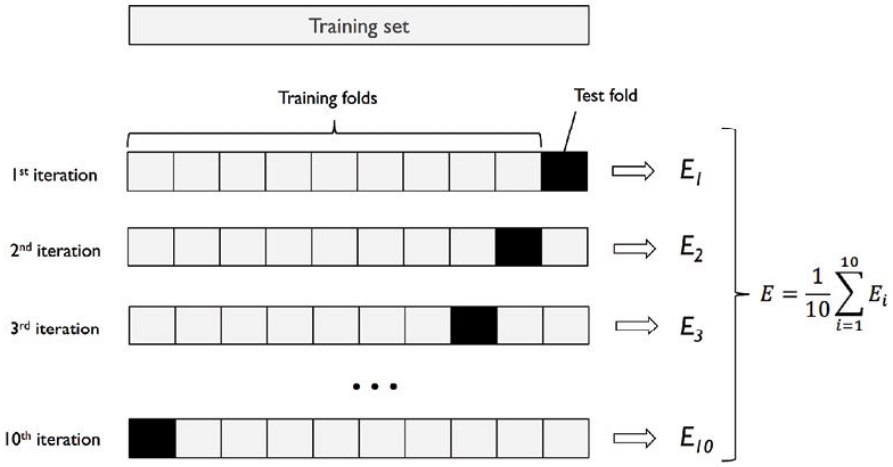


Abbildung 17: Konzept hinter der K-fachen Kreuzvalidierung mit K=10 [75]

Je nach Struktur des Datensatzes werden verschiedene Arten der k-fachen Kreuzvalidierung verwendet. Stratifiziertes K-Fold ist eine erweiterte Version der K-Fold-Kreuzvalidierung, die hauptsächlich für unausgeglichene Datensätze verwendet wird. Genau wie bei K-Fold wird der gesamte Datensatz in gleich große K-Folds aufgeteilt. Allerdings hat bei dieser Technik jede Faltung den gleichen Anteil an Stichproben der Zielgröße wie im gesamten Datensatz [76,77].

In dieser Studie wurde aufgrund der Unausgewogenheit des Datensatzes die „Skfold Cross-Validation“ verwendet. Kreuzvalidierung und andere Parameter werden mit Hilfe eines Tools namens "GridSearchCV" implementiert.

4.5.2 GridSearchCV

GridSearchCV führt eine umfassende Suche über bestimmte Parameterwerte für einen Schätzer durch. GridSearchCV implementiert eine „fit“- und eine „score“-Methode. Die Parameter des Schätzers, der zur Anwendung dieser Methode verwendet wird, werden durch kreuzvalidierte Gittersuche über ein Parametergitter optimiert. Die besten Parameterwerte werden extrahiert und dann die Vorhersagen getroffen. Diese Parameter werden unten erklärt. [78]

„C“ und „Gamma“ sind die Hyperparameter, die die Leistung des SVM-Modells bestimmen. Ausreißer werden mit Hilfe von Fehlergewicht C entsprechend dem Größer der Fehlklassifikation reguliert. Dadurch wird zwischen ein großer Bereich mit vielen oder signifikanten Fehlern und ein kleiner Bereich mit wenigen oder unbedeutenden Fehlern ausgeglichen. Der Fehlklassifizierungs- oder Fehlerterm sagt der SVM-Optimierung, wie viel Fehler erträglich ist. Auf diese Weise kann man den Kompromiss zwischen Entscheidungsgrenze und Fehlklassifizierungsbegriff kontrollieren [79] . Die Grafik über einige Werte von C ist unten (Abbildung 18) angegeben. Wenn C hoch ist, werden alle Datenpunkte korrekt klassifiziert, außerdem besteht die Möglichkeit, dass sie überangepasst werden [68]

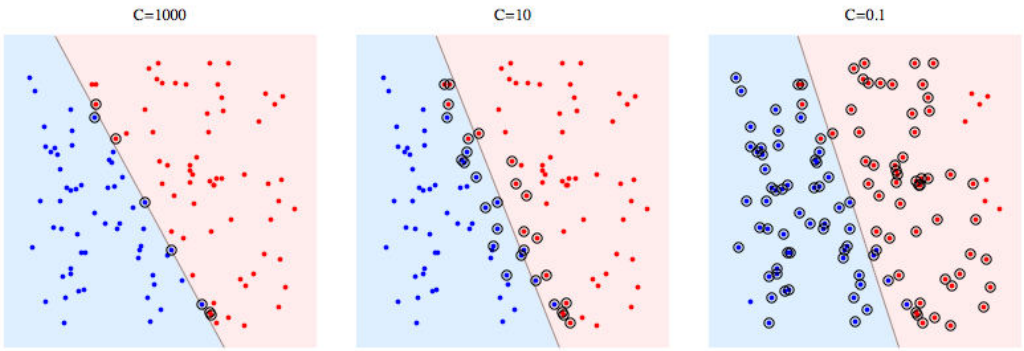


Abbildung 18: C -Werte [79]

Gamma definiert, wie stark die vernünftige Trennlinie die Berechnung beeinflusst. Je höher das Gamma, desto größer der Einfluss benachbarter Punkte während Niedriges Gamma bedeutet, dass auch entfernte Punkte berücksichtigt werden, um die Entscheidungsgrenze zu erhalten d.h. hohes Gamma (a): nur nahe Punkte werden berücksichtigt. Niedriges Gamma (b): auch weit entfernte Punkte werden berücksichtig (Abbildung 19).

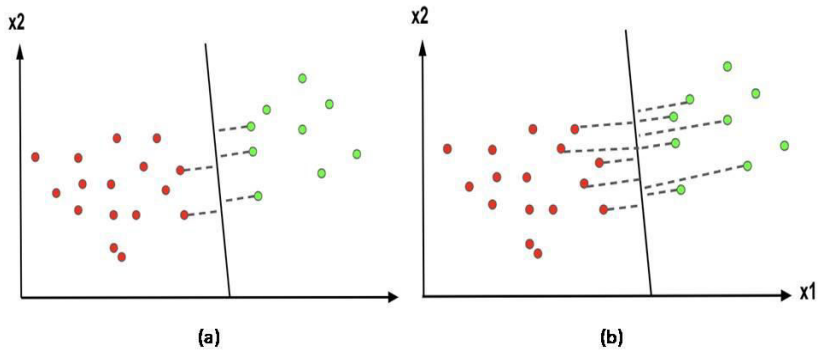


Abbildung 19: Hohes Gamma (a) und Niedriges Gamma (b) [79]

Der "degree" Parameter steuert die Flexibilität der Entscheidungsgrenze. Kerne mit höherer Ordnung stellen eine flexiblere Entscheidungsgrenze bereit [80].

Die SVM wurde ursprünglich für Zwei-Klassen-Probleme entworfen. Daher erfordert die Verwendung von SVM für Mehrklassenprobleme spezielle Ansätze. "OvR" (One vs Rest) und "OvO" (One vs One) sind die beiden am meisten bevorzugten Methoden für Klassifikationsprobleme mit mehreren Klassen.

„One vs Rest“ (auch One-versus-All) bestimmt das Ergebnis anhand mehrerer binärer Entscheidungen [81–83]. Jede Entscheidung wird getroffen, indem eine Klasse gegen alle anderen Klassen bewertet wird. Daraus ergibt sich die Anzahl der benötigten Trennebenen Z:

$$Z = U \tag{4.1}$$

wobei U die Anzahl der Klassen darstellt. Die Lösung eines Drei-Klassen-Problems mit dieser Methode ist schematisch in Abbildung 20 dargestellt. Es wird die „Winner takes all“-Strategie verwendet; Bei dieser Strategie gewinnt die Klasse, die im Klassenvergleich einmal gewinnt, die anderen.

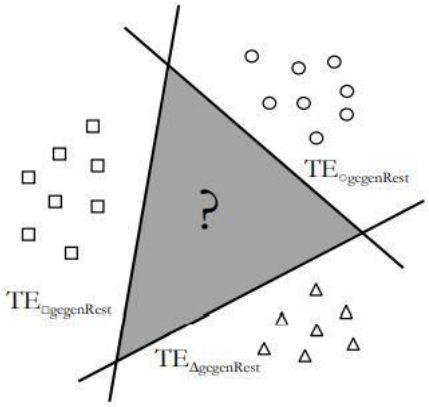


Abbildung 20: One-vs-All-Methode [84]

Der Vorteil dieser Methode liegt in der überschaubaren Anzahl an binären Entscheidungen, die getroffen werden müssen. Allerdings ist der Bereich, den das Modell nicht nutzen kann, eine bestimmte Klasse zu kategorisieren, sehr groß ist (Graubereich). In diesem Fall ist die Zuordnung zu einer Klasse zufällig.

Wie der Namen schon sagt, wird bei der One-vs-One-Klassifizierung jede Klasse paarweise mit der anderen verglichen und eine Entscheidung getroffen. Somit sind die Trennebenen Z bestimmt [81,82]. Wie viele Trennebenen erforderlich sind, wird durch die folgende Gleichung berechnet:

$$Z = \frac{U(U-1)}{2} \tag{4.2}$$

Die Lösung des gleichen Drei-Klassen-Problems aus Abbildung 20 erfolgt damit wie in Abbildung 21 dargestellt.

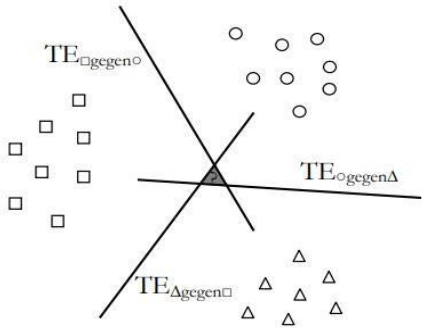


Abbildung 21: One-vs-One-Methode [84]

Hierbei findet die „max wins"-Strategie Anwendung. Als Ergebnis wird die Klasse mit den meisten Entscheidungen ausgewählt. Der Vorteil dieses Verfahrens ist eine kleinere Grauzone, während der Nachteil darin besteht, dass es eine erhöhte Anzahl von Trennebenen erzeugt. Daher hat es eine längere Trainings- und Klassifizierungszeit [83].

Die Tabelle 4 zeigt das beste Ergebnis, das mit 10-facher Stratifiziertes Kreuzvalidierung erzielt wurde. Dementsprechend lieferte der RBF-Kernel unter den angegebenen Parametern die höchsten Ergebnisse (Best_score) für das Teilmenge-Wölbung (Tabelle 4) (siehe Anhang B für Details).

Teilmenge	Kernel	best score	C	Gamma	decision function shape	degree
Mittelwert	Polynomial	0.80	100	100	ovo	1
Schiefe	RBF	0.74	100	10	ovo	1
Wölbung	RBF	0.90	100	1	ovo	1

Tabelle 4: Best "Cross Validation Accuracy" für Teilmengen

4.6. Evaluationskriterien

Das genaue Messen und Bewerten des Erfolgs des Modells ist für eine Aufgabe des ML von entscheidender Bedeutung. Durch die korrekte Bewertung der ausgewählten Metriken wird es einfacher, Verbesserungen am Modell vorzunehmen und daran zu arbeiten, Vorhersagen zu erstellen, die der Realität so nahe kommen. In dieser Richtung ist es notwendig, Begriffe wie „Classification report“ und „Confusion matrix“ zu definieren. Die von GridSearchCV erhaltenen Genauigkeitswerte können über einen "Klassifizierungsbericht" mit der Genauigkeit im Testset verglichen werden. Auswertungen zu diesem Vergleich werden in Kapitel 5 noch einmal erwähnt. Die folgenden Tabelle 5 zeigen die Klassifizierungsberichte der Untergruppen.

Klassifizierungsbericht für Mittelwert	Accuracy score 0.8125 Classification report precision recall f1-score support B 0.90 0.82 0.86 11 S 1.00 0.70 0.82 10 Sch 0.67 0.91 0.77 11 accuracy 0.81 32 macro avg 0.86 0.81 0.82 32 weighted avg 0.85 0.81 0.82 32
Klassifizierungsbericht für Schiefe	Accuracy score 0.6875 Classification report precision recall f1-score support B 0.91 0.91 0.91 11 S 0.56 0.90 0.69 10 Sch 0.60 0.27 0.37 11 accuracy 0.69 32 macro avg 0.69 0.69 0.66 32 weighted avg 0.69 0.69 0.66 32
Klassifizierungsbericht für Wölbung	Accuracy score 0.90625 Classification report precision recall f1-score support B 0.92 1.00 0.96 11 S 1.00 0.70 0.82 10 Sch 0.85 1.00 0.92 11 accuracy 0.91 32 macro avg 0.92 0.90 0.90 32 weighted avg 0.92 0.91 0.90 32

Tabelle 5: Klassifizierungsbericht nach Testdaten in Teilmengen

Precision ist ein Maß dafür, wie viele der getroffenen positiven Vorhersagen korrekt sind (True Positives), d.h. es ist die Gesamtzahl der positiv vorhergesagten Einheiten von True Positive-Elementen dividiert durch die Gesamtzahl der positiv vorhergesagten Einheiten (Spaltensumme der vorhergesagten positiven Ergebnisse [85]. Die Formel dafür lautet:

$$Precision = \frac{TP}{TP + FP} \tag{4.3}$$

True Positive (TP) sind Items, die vom Modell als positiv gekennzeichnet werden und tatsächlich positiv sind, während False Positive(FP) Items sind, die vom Modell als positiv gekennzeichnet werden, aber tatsächlich negativ sind [85]. Precision bezieht sich auf das Verhältnis von Einheiten, die laut Modell positiv sind und tatsächlich positiv sind. Mit anderen Worten, Precision gibt an, wie sehr man dem Modell vertrauen kann, wenn es vorhersagt, dass eine Vorhersage positiv sein wird [85].

Recall ist der Bruchteil der richtig positiven Elemente (TP) dividiert durch die Gesamtzahl der positiv klassifizierten Einheiten (Zeilensumme der tatsächlichen Positiven). False Negative (FN) sind insbesondere die Elemente, die vom Modell als negativ bezeichnet wurden, aber eigentlich positiv sind [85]. Recall misst die Vorhersagegenauigkeit des Modells für die positive Klasse: Er misst intuitiv die Fähigkeit des Modells, alle positiven Einheiten im Datensatz zu finden [85].

$$Recall = \frac{TP}{TP + FN} \tag{4.4}$$

F1-Score bewertet die Leistung des Klassifikationsmodells ausgehend von der Konfusionsmatrix, es aggregiert Precission- und Recall unter dem Konzept des harmonischen Mittelwerts [85]. Der relative Beitrag von Precision und Recall zum F1-Score ist gleich und das harmonische Mittel ist nützlich, um den besten Kompromiss zwischen den beiden Größen zu finden [85].

$$F1 - Score = 2 * \left(\frac{precision * recall}{precision + recall}\right) \tag{4.5}$$

Die Genauigkeit (Accuracy) ist eine der beliebtesten Metriken bei der Mehrklassenklassifizierung und wird direkt aus der Konfusionsmatrix berechnet [85].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.6}$$

Die Genauigkeitsformel berücksichtigt die Summe der True Positive- und True Negative-Elemente im Zähler und die Summe aller Einträge der Konfusionsmatrix im Nenner.

True Positives und True Negatives sind Items, die vom Modell korrekt klassifiziert wurden und sich in der Hauptdiagonale der Konfusionsmatrix befinden, während der Nenner alle Items außer der Hauptdiagonale berücksichtigt, die vom Modell falsch klassifiziert wurden [85].

Beim statistischen Klassifizierungsproblem wird eine Konfusionsmatrix, auch Fehlermatrix genannt, verwendet [36]. Ziel ist es den Erfolg eines Algorithmus in einer Tabelle zu visualisieren. Jede Zeile der Matrix stellt die Instanzen in einer tatsächlichen Klasse dar, während jede Spalte die Instanzen in einer vorhergesagten Klasse darstellt, oder umgekehrt [86].

Zum besseren Verständnis der Thematik werden die oben genannten Metriken im Folgenden anhand einer „Komplexitätsmatrix“ schematisch erläutert. (Tabelle 6) [87,88].

		vorhergesagte Werte		
	Klassen	**A**	**B**	**C**
Tatsächliche Werte	**A**	**Zelle 1**	**Zelle 2**	**Zelle 3**
	B	**Zelle 4**	**Zelle 5**	**Zelle 6**
	C	**Zelle 7**	**Zelle 8**	**Zelle 9**

Tabelle 6: Komplexitätsmatrix - Schematisch

z.B. für **Klasse A**;

$$\text{True Positive (TP)} = \text{Zelle } 1, \tag{4.7}$$

$$\text{False Negative (FN)} = \text{Zelle } 2 + \text{Zelle } 3, \tag{4.8}$$

$$\text{False Positive (FP)} = \text{Zelle } 4 + \text{Zelle } 7, \tag{4.9}$$

$$\text{True Negative (TN)} = \text{Zelle } 5 + \text{Zelle } 6 + \text{Zelle } 8 + \text{Zelle } 9. \tag{4.10}$$

In ähnlicher Weise werden für die **Klasse B** wie folgt berechnet;

$$\text{True Positive (TP)} = \text{Zelle } 5, \tag{4.11}$$

$$\text{False Negative (FN)} = \text{Zelle } 4 + \text{Zelle } 6, \tag{4.12}$$

$$\text{False Positive (FP)} = \text{Zelle } 2 + \text{Zelle } 8, \tag{4.13}$$

$$\text{True Negative (TN)} = \text{Zelle } 1 + \text{Zelle } 3 + \text{Zelle } 7 + \text{Zelle } 9. \tag{4.14}$$

und für die **Klasse C** ist;

$$\text{True Positive (TP)} = \text{Zelle } 9, \tag{4.15}$$

$$\text{False Negative (FN)} = \text{Zelle } 3 + \text{Zelle } 6, \tag{4.16}$$

$$\text{False Positive (FP)} = \text{Zelle } 7 + \text{Zelle } 8, \tag{4.17}$$

$$\text{True Negative (TN)} = \text{Zelle } 1 + \text{Zelle } 2 + \text{Zelle } 4 + \text{Zelle } 5. \tag{4.18}$$

5. Auswertung und Ergebnisse der Klassifizierung im Bergbau

Im Allgemeinen machen die Genauigkeitswerte, die nur von GridSearchCV allein erhalten werden, nicht viel Sinn. Andere Metriken im Klassifizierungsbericht und Best_score- und Genauigkeitswerte sollten ebenfalls verglichen werden [78].

Wie bereits erwähnt, berücksichtigt GridSearchCV nicht den Testset; Die gemeldete Punktzahl stammt aus der Kreuzvalidierung im Trainingssatz. Z.B. ist der 'best_score' für Teilmenge Wölbung 0,90. Dieser Wert ist der Durchschnittswert der Punktzahl in jeder der CV-Falten [78].

„Accuracy_score“ hingegen stammt aus dem Testset. Dies ist die Anwendung der in der Kreuzvalidierung erhaltenen Trainingssätze auf den Testsatz [78].

Andererseits sollten sich diese beiden Werte logischerweise nicht stark unterscheiden, vorausgesetzt, dass sowohl das „CV-Verfahren“ als auch der Trainingstestteil korrekt durchgeführt werden und der Datensatz ausgeglichen ist, was im Wölbung-Datensatz angegeben ist.

Eine Modifikation von Tabelle 4 wird erhalten, indem „accuracy_score“-Werte zu den Teilmengen in der folgenden Tabelle 7 hinzugefügt werden.

Teilmenge	Kernel	best score	accuracy score	C	Gamma	decision function shape	degree
Mittelwert	Poly	0.80	0.81	100	100	ovo	1
Schiefe	RBF	0.74	0.69	100	10	ovo	1
Wölbung	RBF	0.90	0.90	100	1	ovo	1

Tabelle 7: best_score vs accuracy_score

In der „Teilmenge Wölbung“ schnitt der „RBF-Kernel“ am besten ab. Es gibt fast keinen Unterschied zwischen den Werten „Beste Score“ und „ Accuracy Score“. Wie aus dem Klassifizierungsbericht hervorgeht, ist die Verteilung des Testdatensatzes recht ausgeglichen (Tabelle 8: rote Markierung).

Accuracy score 0.90625
Classification report

	precision	recall	f1-score	support
B	0.92	1.00	0.96	11
S	1.00	0.70	0.82	10
Sch	0.85	1.00	0.92	11
accuracy			0.91	32
macro avg	0.92	0.90	0.90	32
weighted avg	0.92	0.91	0.90	32

Tabelle 8: Klassifizierungsbericht für Wölbung

Die folgende Abbildung 22 zeigt die Komplexitätsmatrix des Algorithmus für die Teilmenge Wölbung.

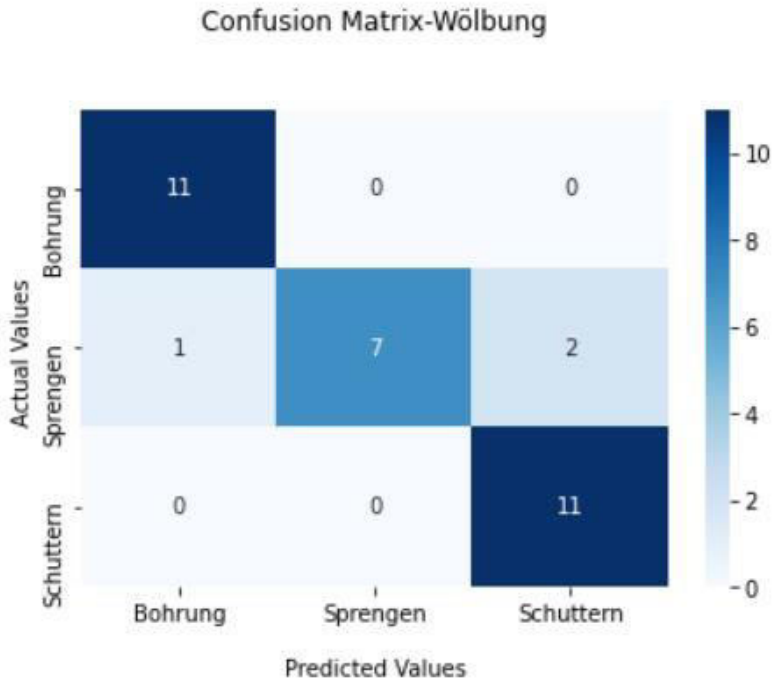


Abbildung 22: Konfusionmatrix nach bestem Ergebnis

Beispielsweise werden die Metriken (Tabelle 8: blaue Markierung) im Klassifizierungsbericht für die Klassen von Bohrung, Sprengen-und Schuttern mit Hilfe der Konfusionsmatrix (Abbildung 15) wie folgt berechnet (Tabelle9);

Klassen	vier mögliche Fälle	Berechnung den Metriken
Bohrung	True Positive (TP) = 11, False Negative (FN) = 0 False Positive (FP) = 1 True Negative (TN) = 0 + 7 +2 +11 = 20	$Precision = \frac{11}{11+1} = 0.916$ $Recall = \frac{11}{11+0} = 1$ $F1 - Score = 2 * \left(\frac{0.916 * 1}{0.916 + 1}\right) = 0.956$ $Accuracy = \left(\frac{11+20}{11+0+1+20}\right) = 0.968$
Sprengen	True Positive (TP) = 7 False Negative (FN) = 1 + 2 = 3 False Positive (FP) = 0 True Negative (TN) = 11 + 0 +0 + 11 = 22	$Precision = \frac{7}{7+0} = 1$ $Recall = \frac{7}{7+3} = 0.7$ $F1 - Score = 2 * \left(\frac{1 * 0.7}{1 + 0.7}\right) = 0.823$ $Accuracy = \left(\frac{7+22}{7+3+0+22}\right) = 0.906$
Schuttern	True Positive (TP) = 11 False Negative (FN) = 0 False Positive (FP) = 0 + 2 = 2 True Negative (TN) = 11 + 0 + 1 + 7 = 19	$Precision = \frac{11}{11+2} = 0.846$ $Recall = \frac{11}{11+0} = 1$ $F1 - Score = 2 * \left(\frac{0.846 * 1}{0.846 + 1}\right) = 0.916$ $Accuracy = \left(\frac{11+19}{11+0+2+19}\right) = 0.937$

Tabelle 9: Beurteilung des Klassifikators im Teilmenge Wölbung

Wenn ein Fall positiv ist und positiv vorhergesagt wird, ist er "True Positiv". Alle Bohrungen und Schuttern im Testsatz und 7 Testdaten des Sprengens wurden korrekt vorhergesagt. Wenn ein Fall negativ ist und als negativ vorhergesagt wird, ist er "True Negativ". 20 Testdaten beim Bohrung, 22

beim Sprengen und 19 beim Schuttern wurden als "True Negativ" bewertet. Ein Testdaten für die Bohrung wird als Bohrung vorhergesagt, obwohl es keine Bohrung war, d.h. wenn ein Fall negativ ist, aber als positiv vorhergesagt wird, ist ein Falsch Positiv. Mit anderen Worten, diese ist "False Negative" für die Klasse-Sprengen, weil drei Mustern von Sprengen (eine für Bohrung, zwei für Schuttern) in den falschen Klassen vorhergesagt wurden, also wenn ein Fall positiv ist, aber negativ vorhergesagt wird ist es Falsch Negativ.

Die Metriken werden mit Hilfe der oben beschriebenen „Fälle“ berechnet. Wie bereits erwähnt, bewertet F1-Score die Leistung des Klassifizierungsmodells basierend auf der Konfusionsmatrix. Genauigkeit (Accuracy) wird verwendet, wenn True Positives und True Negatives wichtiger sind, während F1-Score verwendet wird, wenn False Negatives und False Positives sehr wichtig sind. Während Accuracy verwendet werden kann, wenn die Klassenverteilung ähnlich ist, ist die F1-score eine bessere Metrik, wenn es unausgewogene Klassen gibt [89]. Da die Verteilung im Testset gut war und der F1-Score und die Genauigkeitswerte sehr nahe beieinander lagen, wurde der Genauigkeitswert in der Auswertungsphase verwendet.

Um die Gesamtgenauigkeit zu berechnen, wird die Summe der True Positiv-Werte jeder Klasse (Abbildung 23: rote Markierung) auf der Konfusionsmatrix durch die Gesamtzahl der Testdaten dividiert;

$$Accuracy = \frac{11 + 7 + 11}{32} = 0.90625 \tag{4.19}$$

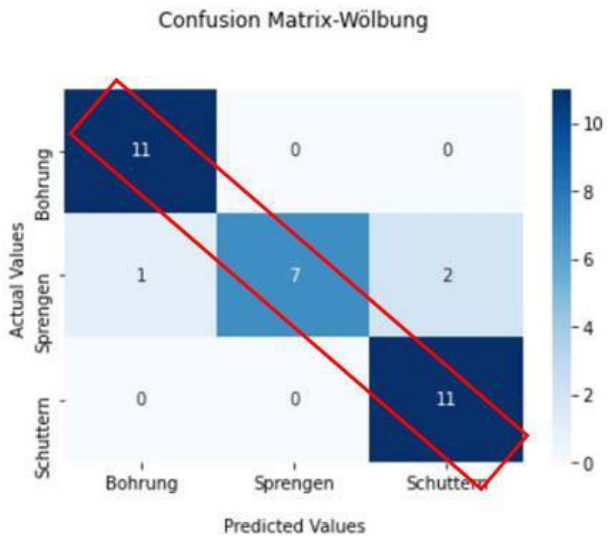


Abbildung 23: True Positiv Werte

5.1. Ergebnisse der Klassifizierung

RBF-Kernel liefert das erfolgreichste und zuverlässigste Klassifikationsergebnis für die Teilmenge-Wölbung mit den angegebenen Parametern (Tabelle 10) und diese wurde durch andere Parameter wie Precision, F1-Score und Recall gestützt.

Teilmenge	Kernel	best score	accuracy score	C	Gamma	decision function shape	degree
Wölbung	RBF	0.90	0.90	100	1	ovo	1

Tabelle 10: Bestes Ergebnis

Die Klassifizierung kann je nach Genauigkeitswert in mehrere Gruppen eingeteilt werden (Tabelle 11) [90].

0,90 - 1,00	=	sehr gute Klassifizierung
0,80 - 0,90	=	gute Klassifizierung
0,70 - 0,80	=	ausreichende Klassifizierung
0,60 - 0,70	=	schlechte Klassifizierung
0,50 - 0,60	=	falsche Klassifizierung

Tabelle 11: Klassifizierungsgruppen [90]

Dementsprechend gehört die Genauigkeit des Test-Sets mit einem Genauigkeitswert von 0,90 zur Gruppe „gute Klassifizierung“.

Als Ergebnis wurde ein Drei-Klassen-Klassifizierungsproblem von Prozessdaten aus dem Bergbau mit einer SVM gelöst. Die mit den SVM-Kernel-Algorithmen erhaltenen Ergebnisse können in vielen in Abschnitt 2.2 erwähnten Bergbau Aufgaben verwendet werden.

6. Zusammenfassung und Ausblick

In dieser Masterarbeit wird ein auf SVMs basierender Klassifikationsansatz zur Klassifikation und Detektion von Ereignissen im Bergbau vorgeschlagen.

Zunächst wurden Teilmengen aus Zeitreihendaten unter Verwendung klassischer statistischer Parameter erstellt. Nach Sicherstellung der Datenkonsistenz wurden die SVM-Algorithmen mit dem Kreuzvalidierungsverfahren validiert und das höchste und zuverlässigste Klassifizierungsergebnis erzielt. Somit wurde mit Hilfe von SVM-Kernelfunktionen ein Drei-Klassen-Klassifikationsproblem gelöst.

Insbesondere sind einige zusätzliche Verbesserungen erforderlich, um den Datenvorverarbeitungsprozess und die Klassifizierung effektiver und effizienter zu gestalten. Aus Anwendersicht erscheint eine weitergehende Klassifizierung mit Software- und Hardwareoptimierung durchaus möglich. Probleme der Größe und Unausgeglichenheit von Daten können einfach und weitgehend mit einer Teilmenge von Datensätzen und der optimalen Anzahl von Merkmalen gelöst werden, und es können bessere Schätzungen erhalten werden.

Darüber hinaus kann der Klassifikations- und Vorhersageerfolg (vergleichsweise) weiter verbessert werden, indem andere ML-verfahren (Long Short Term Memory, Artificial Neural Networks, Logistic Regression usw.) angewendet werden, die für die Struktur des Datensatzes(Signal Analysis) geeignet sind. Alle genannten Maßnahmen stellen daher einen erheblichen Rechenaufwand dar und erfordern daher eine Optimierung der zugrunde liegenden Prozesse.

Abschließend bleibt zu sagen, dass die Erstellung verschiedener Datenteilmengen und der Vergleich dieser Teilmengen unter Verwendung von SVM-Kernalgorithmen, gute Klassifizierungs- und Genauigkeitsergebnisse erzielten. Außerdem kann das Verfahren in analoger Form mit anderen Signalwerten in anderen Branchen eingesetzt werden.

Abbildungsverzeichnis

Abbildung 1: Überanpassung, Unteranpassung und Fehlerrate-Datensätze Diagramm [8,17] 7
Abbildung 2: Mögliche Hyperebenen (a) und optimal Hyperebene (b) [21] 9
Abbildung 3: nicht-lineare Datentrennung [23] 10
Abbildung 4: Optimal getrennte Hyperebene über drei Stützvektoren[24] 12
Abbildung 5: Drift der Hyperebene aufgrund der Optimierungsbedingungen [27] 13
Abbildung 6: Anwendung einer Soft-Margin bei nicht-trennbaren Trainingsdaten [24] 16
Abbildung 7: Lineare Trennung durch Transformation in einen höher dimensionalen Raum [30] 19
Abbildung 8: Zeitreihenkomponenten: (a)Trend; (b) Saisonal; (c) Zyklisch; (d) Unregelmäßig [35] 23
Abbildung 9: Zeitreihen für Sprengen (oben), Bohrung (mittig) und Schuttern (unten) 26
Abbildung 10: „Bohrung“-Werte gemessen zwischen 00:28:36-01:42:22 Uhr am 22.11.2018 27
Abbildung 11: Arten von Schiefen [48] 28
Abbildung 12: Art von Kurtosis [49] 29
Abbildung 13: Teilmenge Mittelwert 30
Abbildung 14: Schiefe, Mittelwert und Wölbung für Teilmenge-Sprengen 31
Abbildung 15: Schiefe, Mittelwert und Wölbung für Teilmenge-Bohrung 32
Abbildung 16: Schiefe, Mittelwert und Wölbung für Teilmenge-Schiefe 33
Abbildung 17: Konzept hinter der K-fachen Kreuzvalidierung mit K=10 [64] 38
Abbildung 18: C -Werte [68] 39
Abbildung 19: Hohes Gamma (a) und Niedriges Gamma (b) [68] 40
Abbildung 20: One-vs-All-Methode [73] 41
Abbildung 21: One-vs-One-Methode [73] 42
Abbildung 22: Konfusionmatrix nach bestem Ergebnis 48
Abbildung 23: True Positiv Werte 51

Tabellenverzeichnis

Tabelle 1: Ergebnisse von Kernelfunktionen für Mittelwert 36
Tabelle 2: Ergebnisse von Kernelfunktionen für Schiefe 36
Tabelle 3: Ergebnisse von Kernelfunktionen für Wölbung 36
Tabelle 4: Best “Cross Validation Accuracy” für Teilmengen 42
Tabelle 5: Klassifizierungsbericht nach Testdaten in Teilmengen 43
Tabelle 6: Komplexitätsmatrix - Schematisch 45
Tabelle 7: best_score vs accuracy_score 47
Tabelle 8: Klassifizierungsbericht für Wölbung 48
Tabelle 9: Beurteilung des Klassifikators im Teilmenge Wölbung 49
Tabelle 10: Genauigkeitswertgruppen [79] 51
Tabelle 11: Bestes Ergebnis 51

Literaturverzeichnis

1. Chang, C.-C.; Lin, C.-J. LIBSVM. *ACM Trans. Intell. Syst. Technol.* **2011**, *2*, 1–27, doi:10.1145/1961189.1961199.
2. Huang, B. *Comprehensive GIS*: *Geographic Information System;* Elsevier: San Diego, 2017, ISBN 978-0-12-804793-4.
3. Iqbal, T.; Elahi, A.; Wijns, W.; Shahzad, A. Exploring Unsupervised Machine Learning Classification Methods for Physiological Stress Detection. *Front. Med. Technol.* **2022**, *4*, 782756, doi:10.3389/fmedt.2022.782756.
4. BISHOP, C.M. *PATTERN RECOGNITION AND MACHINE LEARNING;* SPRINGER-VERLAG NEW YORK, 2016, ISBN 978-1-4939-3843-8.
5. Géron, A. *HANDS-ON MACHINE LEARNING WITH SCIKIT-LEARN, KERAS, AND TENSORFLOW: CONCEPTS, TOOLS, AND TECHNIQUES;* O'REILLY MEDIA: SEBASTOPOL, 2019, ISBN 9781492032649.
6. Cortes, C.; Vapnik, V. Support-vector networks. *Mach Learn* **1995**, *20*, 273–297, doi:10.1007/BF00994018.
7. Boser, B.E.; Guyon, I.M.; Vapnik, V.N. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory - COLT '92.* the fifth annual workshop, Pittsburgh, Pennsylvania, United States, 27–29 Jul. 1992; Haussler, D., Ed.; ACM Press: New York, New York, USA, 1992; pp 144–152, ISBN 089791497X.
8. Kumar, N. Advantages and Disadvantages of SVM (Support Vector Machine) in Machine Learning. Available online: http://theprofessionalspoint.blogspot.com/2019/03/advantages-and-disadvantages-of-svm.html (accessed on 19 July 2022).
9. Navulur, K. *Multispectral image analysis using the object-oriented paradigm;* CRC Press: Boca Raton, 2020, ISBN 9780367446246.
10. Vezhapparambu, V.; Eidsvik, J.; Ellefmo, S. Rock Classification Using Multivariate Analysis of Measurement While Drilling Data: Towards a Better Sampling Strategy. *Minerals* **2018**, *8*, 384, doi:10.3390/min8090384.
11. Navarro, J.; Seidl, T.; Hartlieb, P.; Sanchidrián, J.A.; Segarra, P.; Couceiro, P.; Schimek, P.; Godoy, C. Blastability and Ore Grade Assessment from Drill Monitoring for Open Pit Applications. *Rock Mech Rock Eng* **2021**, *54*, 3209–3228, doi:10.1007/s00603-020-02354-2.
12. Peng, P.; He, Z.; Wang, L.; Jiang, Y. Automatic Classification of Microseismic Records in Underground Mining: A Deep Learning Approach. *IEEE Access* **2020**, *8*, 17863–17876, doi:10.1109/ACCESS.2020.2967121.
13. Kortström, J.; Uski, M.; Tiira, T. Automatic classification of seismic events within a regional seismograph network. *Computers & Geosciences* **2016**, *87*, 22–30, doi:10.1016/j.cageo.2015.11.006.
14. Vallejos, J.A.; McKinnon, S.D. Logistic regression and neural network classification of seismic records. *International Journal of Rock Mechanics and Mining Sciences* **2013**, *62*, 86–95, doi:10.1016/j.ijrmms.2013.04.005.
15. Dong, L.; Wesseloo, J.; Potvin, Y.; Li, X. Discrimination of Mine Seismic Events and Blasts Using the Fisher Classifier, Naive Bayesian Classifier and Logistic Regression. *Rock Mech Rock Eng* **2016**, *49*, 183–211, doi:10.1007/s00603-015-0733-y.
16. Shang, X.; Li, X.; Morales-Esteban, A.; Chen, G. Improving microseismic event and quarry blast classification using Artificial Neural Networks based on Principal Component Analysis. *Soil Dynamics and Earthquake Engineering* **2017**, *99*, 142–149, doi:10.1016/j.soildyn.2017.05.008.

17. Budakoğlu, E.; Horasan, G. Classification of seismic events using linear discriminant function (LDF) in the Sakarya region, Turkey. *Acta Geophys.* **2018**, *66*, 895–906, doi:10.1007/s11600-018-0179-1.
18. Poole, D.; Mackworth, A.; Goebel, R.; Poole, D.L.; Mackworth, A.K.; Goebel, R.G. *Computational intelligence*: *A logical approach;* Oxford University Press: New York, 1998, ISBN 9780195102703.
19. Bavakutty, M. *Research on library computerisation;* Ess Ess Publ: New Delhi, 2006, ISBN 9788170004783.
20. Ferber, R. *Information retrieval*: *Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web;* Dpunkt Verlag: Heidelberg, 2003, ISBN 9783898642132.
21. Graf, R.J. Objektklassifizierung mit Support Vector Machines. Available online: http://unigis.sbg.ac.at/files/Mastertheses/Full/1436.pdf (accessed on 15 October 2021).
22. TONNIES, K.D. *GRUNDLAGEN DER BILDVERARBEITUNG;* PEARSON STUDIUM: [Place of publication not identified], 2005, ISBN 978-3-86326-637-0.
23. Handels, H. *Medizinische Bildverarbeitung*: *Bildanalyse, Mustererkennung und Visualisierung für die computergestützte ärztliche Diagnostik und Therapie | Studium*, 2., überarbeitete und erweiterte Auflage; Vieweg+Teubner: Wiesbaden, 2009, ISBN 978-3-8351-0077-0.
24. Hermes, T. *Digitale Bildverarbeitung*: *Eine praktische Einführung;* Hanser: München, 2005, ISBN 9783446229693.
25. Niemann, H. *Klassifikation von Mustern;* Springer Berlin Heidelberg: Berlin, Heidelberg, 1983, ISBN 978-3-642-47517-7.
26. Guyon, I. *Feature extraction*: *Foundations and applications;* Springer-Verlag, ISBN 978-3-540-35488-8.
27. Lange, N. *Geoinformatik;* Springer Berlin Heidelberg, 2006, ISBN 978-3540282914.
28. Backhaus, K.; Erichson, B.; Gensler, S.; Weiber, R.; Weiber, T. *Multivariate Analysemethoden*: *Eine anwendungsorientierte Einführung*, 16., vollständig überarbeitete und erweiterte Auflage; Springer Gabler: Wiesbaden, 2021, ISBN 978-3-658-32425-4.
29. Duda, R.O.; Hart, P.E. *Pattern classification and scene analysis*, 27. print; Wiley: New York, 1976, ISBN 978-0471223610.
30. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods;* Cristianini, N.; Shawe-Taylor, J., Eds.; Cambridge University Press, 2013, ISBN 9780521780193.
31. Lohninger, H. Grundlagen der Statistik. Available online: http://www.statistics4u.info/fundstat_germ/ (accessed on 28 July 2022).
32. *Support vector machines*: *Theory and applications;* Wang, L., Ed.; Springer: Berlin, New York, 2005, ISBN 9783540243885.
33. R.Gandhi. Support Vector Machine: Introduction to Machine Learning Algorithms. Available online: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (accessed on 19 July 2022).
34. Ray, S. Understanding Support Vector Machine(SVM). Available online: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/ (accessed on 19 July 2022).
35. Öztürk, M. Python ile Sınıflandırma Analizleri: Destek Vektör Makinası – DVM. Available online: https://miracozturk.com/python-ile-siniflandirma-analizleri-destek-vektor-makinasi-dvm/ (accessed on 19 July 2022).
36. Abe, S. *Support Vector Machines for Pattern Classification*: *Advances in Pattern Recognition.;* Springer London: London, 2010, ISBN 978-1-84996-097-7.
37. Georgii, H.-O. *Stochastik*, 0005 Expanded and Re; De Gruyter: Berlin/Boston, UNITED STATES, 2015, ISBN 978-3-11-035969-5.

38. Burges, C.J. A Tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery 2. *Data Mining and Knowledge Discovery* **1998**, *2*, 121–167, doi:10.1023/A:1009715923555.
39. *Support Vector Machines - Teil 1: Ein theoretischer Überblick*: *Zeitschrift für Geodäsie, Geoinformation und Landmanagement 3.;* Heinert, M., Ed.; DVW e.V.-Gesellschaft für Geodäsie, Geoinformation und Landmanagement: Vogtsburg-Oberrotweil, Germany, 2010.
40. *Advances in Kernel Methods*: *Support vector learning;* Burges, C.J.; Schölkopf, B.; Smola, A.J., Eds.; MIT Press: Cambridge (MA) [etc.], 1998, ISBN 0262194163.
41. Chiang, A.C.; Wainwright, K. *Fundamental methods of mathematical economics*, 4th ed.; McGraw-Hill: Madrid [etc.], 2005, ISBN 978-0071238236.
42. Pratiwi, K.S. Support Vector Machine Classification with Python. Available online: https://medium.com/@kurniasp/support-vector-machine-classification-with-python-64521fbd5b57 (accessed on 19 July 2022).
43. Steinwart, I.; Christmann, A.; Jordan, M.; Kleinberg, J.; Schölkopf, B. *Support Vector Machines;* Springer Science+Business Media, LLC: New York, NY, 2008, ISBN 978-0-387-77242-4.
44. Vapnik, V. *The Nature of Statistical Learning Theory*, 2nd ed. 2000; Springer New York: New York, NY, 2000, ISBN 978-1-4757-3264-1.
45. Hipel, K.W.; McLeod, A.I. *Time series modelling of water resources and environmental systems;* Elsevier: Amsterdam, New York, 1994, ISBN 9780080870366.
46. Raicharoen, T.; Lursinsap, C.; Sanguanbhokai, P. Application of critical support vector machine to time series prediction. In *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.* ISCAS 2003. International Symposium on Circuits and Systems, Bangkok, Thailand, 25-28 May 2003; IEEE, 2003; V-741-V-744, ISBN 0-7803-7761-3.
47. Ciaburro, G.; Iannace, G. Machine Learning-Based Algorithms to Knowledge Extraction from Time Series Data: A Review. *Data* **2021**, *6*, 55, doi:10.3390/data6060055.
48. Adhikari, R.; Agrawal, R.K. An Introductory Study on Time Series Modeling and Forecasting **2013**, doi:10.48550/arXiv.1302.6613.
49. Zhang, G. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing* **2003**, *50*, 159–175, doi:10.1016/S0925-2312(01)00702-0.
50. Zhang, G.P. A neural network ensemble method with jittered training data for time series forecasting. *Information Sciences* **2007**, *177*, 5329–5346, doi:10.1016/j.ins.2007.06.015.
51. Python. What is Python? Available online: https://docs.python.org/3/faq/general.html#what-is-python (accessed on 19 July 2022).
52. Python. Design and History Python. Available online: Design and History FAQ — Python 3.10.5 documentation (accessed on 19 July 2022).
53. Hunter, J.D. Matplotlib: A 2D Graphics Environment. *Comput. Sci. Eng.* **2007**, *9*, 90–95, doi:10.1109/MCSE.2007.55.
54. NumPy. NumPy. Available online: https://numpy.org/doc/stable/user/whatisnumpy.html (accessed on 19 July 2022).
55. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Müller, A.; Nothman, J.; Louppe, G.; et al. Scikit-learn: Machine Learning in Python **2012**, doi:10.48550/arXiv.1201.0490.
56. Pandas. Pandas. Available online: https://www.python-kurs.eu/pandas.php (accessed on 19 July 2022).
57. seaborn.pydata.org. Available online: https://seaborn.pydata.org/ (accessed on 14 August 2022).

58. Laerd Statistic. Measures of Central Tendency. Available online: https://statistics.laerd.com/statistical-guides/measures-central-tendency-mean-mode-median.php (accessed on 16 August 2022).
59. National Institute of Standards and Technology, NIST. Measures of Skewness and Kurtosis. Available online: https://www.itl.nist.gov/div898/handbook/eda/section3/eda35b.htm (accessed on 16 August 2022).
60. Dugar, D. Skew and Kurtosis: 2 Important Statistics terms you need to know in Data Science. Available online: https://codeburst.io/2-important-statistics-terms-you-need-to-know-in-data-science-skewness-and-kurtosis-388fef94eeaa (accessed on 16 August 2022).
61. Gawali, S. Shape of data: Skewness and Kurtosis. Available online: https://www.analyticsvidhya.com/blog/2021/05/shape-of-data-skewness-and-kurtosis/ (accessed on 16 August 2022).
62. Ron, K. "Glossary of terms"- Machine Learning. *Mach Learn* **1998**, *30*, 5–6, doi:10.1023/A:1007411609915.
63. Ripley, B.D. *Pattern Recognition and Neural Networks;* Cambridge University Press, 2014, ISBN 9780521460866.
64. Neural Network FAQ, part 1 of 7: Introduction: What are the population, sample, training set, design set, validation set, and test set? Available online: https://www.gbif.es/wp-content/uploads/2010/05/NeuralNetworkFAQ.pdf (accessed on 20 July 2022).
65. Larose, D.T.; Larose, C.D. *Discovering knowledge in data*: *An introduction to data mining*, 2nd ed.; Wiley: Hoboken, 2014, ISBN 978-0-470-90874-7.
66. Xu, Y.; Goodacre, R. On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning. *J. Anal. Test.* **2018**, *2*, 249–262, doi:10.1007/s41664-018-0068-2.
67. Jason Brownlee. What is the Difference Between Test and Validation Datasets?: in Machine Learning Process. Available online: https://machinelearningmastery.com/difference-test-validation-datasets/ (accessed on 20 July 2022).
68. Ron, K. A study of cross-validation and bootstrap for accuracy estimation and model selection. Available online: https://www.researchgate.net/publication/2352264_A_Study_of_Cross-Validation_and_Bootstrap_for_Accuracy_Estimation_and_Model_Selection (accessed on 20 July 2022).
69. Nishesh Gogia. Why Scaling is Important in Machine Learning? Available online: https://medium.com/analytics-vidhya/why-scaling-is-important-in-machine-learning-aee5781d161a (accessed on 10 June 2022).
70. Hsu, C.W., Chang, C.C. & C.J. Lin. A Practical Guide to Support Vector Classification (accessed on 16 June 2022).
71. Jason Brownlee. How to Use StandardScaler and MinMaxScaler Transforms in Python: in Data Preparation. Available online: https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/ (accessed on 9 June 2022).
72. Yugesh Verma. Why Data Scaling is important in Machine Learning & How to effectively do it. Available online: https://analyticsindiamag.com/why-data-scaling-is-important-in-machine-learning-how-to-effectively-do-it/ (accessed on 10 June 2022).
73. Zakaria Jaadi. When and Why to Standardize Your Data?: A simple guide on when it is necessary to standardize your data. Available online: https://builtin.com/data-science/when-and-why-standardize-your-data (accessed on 16 June 2022).
74. Shawe-Taylor, J.; Cristianini, N. *Kernel Methods for Pattern Analysis;* Cambridge University Press, 2011, ISBN 9780521813976.

75. Raschka, S.; Mirjalili, V. *Python machine learning*: *Machine learning and deep learning with Python, scikit-learn, and TensorFlow*, Second edition, fourth release,[fully revised and updated]; Packt Publishing: Birmingham, Mumbai, 2018, ISBN 9781787125933.
76. Kumar, A. K-Fold Cross Validation. Available online: https://vitalflux.com/k-fold-cross-validation-python-example/ (accessed on 21 July 2022).
77. Brownlee, J. Imbalanced Classification: How to Fix k-Fold Cross-Validation for Imbalanced Classification. Available online: https://machinelearningmastery.com/cross-validation-for-imbalanced-classification/ (accessed on 21 July 2022).
78. sklearn.model_selection.GridSearchCV. Available online: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html (accessed on 1 July 2022).
79. Clare Liu. SVM Hyperparameter Tuning using GridSearchCV. Available online: https://www.vebuso.com/2020/03/svm-hyperparameter-tuning-using-gridsearchcv/ (accessed on 8 July 2022).
80. Ben-Hur, A.; Weston, J. A user's guide to support vector machines. *Methods Mol. Biol.* **2010**, *609*, 223–239, doi:10.1007/978-1-60327-241-4_13.
81. Schölkopf, B.; Smola, A.J. *Learning with kernels*: *Support vector machines, regularization, optimization and beyond;* MIT Press: Cambridge, Mass., London, 2009?, ISBN 0262194759.
82. Hsu, C.-W.; Lin, C.-J. A comparison of methods for multiclass support vector machines. *IEEE Trans. Neural Netw.* **2002**, *13*, 415–425, doi:10.1109/72.991427.
83. MILGRAM, J.; CHERIET, M.; SABOURIN, R. "One Agains One" or "One Against All": Which One is Better for Handwriting Recognition with SVMs?: Tenth International Workshop on Frontiers in Handwriting Recognition. *Information Sciences* **2006**.
84. Dipl.-Ing. (FH) Maria Tromme. Ein Beitrag zur Anwendung von Support-Vektor-Maschinen zur robusten nichtlinearen Klassifikation komplexer biologischer Daten. DISSERTATION; Technischen Universität Ilmenau, 2016.
85. Grandini, M.; Bagli, E.; Visani, G. *Metrics for Multi-Class Classification: an Overview*, 2020.
86. Powers, D.M.W. Journal of Machine Learning Technologies: Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. *J of Mach Lear Tech* **2011**, 37–63, doi:10.9735/2229-3981.
87. Vikas Vyas. Confusion Matrix Understanding Multi-Class Machine Learning Model. Available online: https://medium.com/@vikas.vyas5/confusion-matrix-understanding-multi-class-machine-learning-model-8f2ed0d3c2b3 (accessed on 12 July 2022).
88. Mohajon, J. Confusion Matrix for Your Multi-Class Machine Learning Model. Available online: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826 (accessed on 12 July 2022).
89. Educative. What is the F1-score? Available online: https://www.educative.io/answers/what-is-the-f1-score (accessed on 22 July 2022).
90. Vladimir N. Vapnik. *The Nature of Statistical Learning Theory;* Springer New York: New York, NY, 1995, ISBN 978-1-4757-2440-0.

Anhang

Anhang A: Python Codes-Erstellung von Teilmengen

In [1]:
```
import numpy as np
import pandas as pd
import os
```

In [2]:
```
dir = "D:\\Uni\Diplomarbeit\Diplomarbeit-Daten\Bohrung"   # Ordner mit allen Files für  Bohren - Pfad einfügen
df_skew_B = pd.DataFrame(columns=["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"])
df_mean_B = pd.DataFrame(columns=["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"])
df_kurtosis_B = pd.DataFrame(columns=["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"])
          # Leerer df, in den die mean, skew, kurtosis-Werte gespeichert werden

for path, subdirs, files in os.walk(dir):      # Durchgehen aller files im Verzeichnis dir (geht auch Unterordner durch)
    for name in files:
        filepath = os.path.join(path, name)     # Zusammensetzen des Dateipfades einer einzelnen Datei
        #print(filepath)

        df_file_B = pd.read_csv(filepath, sep=",", header=0)  # Auslesen der csv-Datei
        #print(df_file_B)

        del df_file_B["Unnamed: 0"]
        del df_file_B["t"]

        df_file_mean = df_file_B.mean(axis=0)  # Bildung der Mittelwerte des einzelnen files
        df_file_skew = df_file_B.skew(axis=0)  # Bildung der Schiefe des einzelnen files
        df_file_kurtosis = df_file_B.kurtosis(axis=0)  # Bildung der Wölbung des einzelnen files

        a = 'B'
        df_file_skew['Events']= a
        df_file_mean['Events']= a
        df_file_kurtosis['Events']= a

        df_mean_B = df_mean_B.append(df_file_mean, ignore_index=True) # Speichern der Mittelwerte in den  df
        df_skew_B = df_skew_B.append(df_file_skew, ignore_index=True) # Speichern der Schiefe in den  df
        df_kurtosis_B = df_kurtosis_B.append(df_file_kurtosis, ignore_index=True) # Speichern der Wölbung in den df

#df_mean_B
#df_skew_B
#df_kurtosis_B
```

In [ ]:
```
```

In [6]:
```
dir = "D:\\Uni\Diplomarbeit\Diplomarbeit-Daten\Schuttern"  # Ordner mit allen Files für  Schuttern - Pfad einfügen
df_skew_Sch = pd.DataFrame(columns=["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"])
df_mean_Sch = pd.DataFrame(columns=["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"])
df_kurtosis_Sch = pd.DataFrame(columns=["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"])

for path, subdirs, files in os.walk(dir):      # Durchgehen aller files im Verzeichnis dir (geht auch Unterordner durch)
    for name in files:
        filepath = os.path.join(path, name)     # Zusammensetzen des Dateipfades einer einzelnen Datei

        df_file_Sch = pd.read_csv(filepath, sep=",", header=0)  # Auslesen der csv-Datei

        del df_file_Sch["Unnamed: 0"]
        del df_file_Sch["t"]

        df_file_mean = df_file_Sch.mean(axis=0)    # Bildung der Mittelwerte des einzelnen files
        df_file_skew = df_file_Sch.skew(axis=0)    # Bildung der Schiefe des einzelnen files
        df_file_kurtosis = df_file_Sch.kurtosis(axis=0)  # Bildung der Wölbung des einzelnen files

        b = 'Sch'
        df_file_skew['Events']= b
        df_file_mean['Events']= b
        df_file_kurtosis['Events']= b

        df_mean_Sch = df_mean_Sch.append(df_file_mean, ignore_index=True) # Speichern der Mittelwerte in den df
        df_skew_Sch = df_skew_Sch.append(df_file_skew, ignore_index=True) # Speichern der Schiefe in den df
        df_kurtosis_Sch = df_kurtosis_Sch.append(df_file_kurtosis, ignore_index=True) # Speichern der Wölbung in den df

#df_mean_Sch
#df_skew_Sch
#df_kurtosis_Sch
```

In [10]:
```
dir = "D:\\Uni\Diplomarbeit\Diplomarbeit-Daten\Sprengen"  # Ordner mit allen Files für Sprengen - Pfad einfügen
df_skew_S = pd.DataFrame(columns=["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"])
df_mean_S = pd.DataFrame(columns=["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"])
df_kurtosis_S = pd.DataFrame(columns=["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"])

for path, subdirs, files in os.walk(dir):      # Durchgehen aller files im Verzeichnis dir (geht auch Unterordner durch)
    for name in files:
        filepath = os.path.join(path, name)     # Zusammensetzen des Dateipfades einer einzelnen Datei
```

```
        df_file_S = pd.read_csv(filepath, sep=",", header=0)  # Auslesen der csv-Datei

        del df_file_S["Unnamed: 0"]
        del df_file_S["t"]

        df_file_mean = df_file_S.mean(axis=0)  # Bildung der Mittelwerte des einzelnen files
        df_file_skew = df_file_S.skew(axis=0)  # Bildung der Schiefe des einzelnen files
        df_file_kurtosis = df_file_S.kurtosis(axis=0)  # Bildung der Wölbung des einzelnen files

        c = 'S'
        df_file_skew['Events']= c
        df_file_mean['Events']= c
        df_file_kurtosis['Events']= c

        df_mean_S = df_mean_S.append(df_file_mean, ignore_index=True) # Speichern der Mittelwerte in den df
        df_skew_S = df_skew_S.append(df_file_skew, ignore_index=True) # Speichern der Schiefe in den df
        df_kurtosis_S = df_kurtosis_S.append(df_file_kurtosis, ignore_index=True)# Speichern der Wölbung in den df

#df_mean_S
#df_skew_S
#df_kurtosis_S
```

In [19]:

```
df_mean_all = pd.concat([df_mean_B, df_mean_S, df_mean_Sch ]).reset_index(drop=True)

df_skew_all = pd.concat([df_skew_B, df_skew_S, df_skew_Sch ]).reset_index(drop=True)

df_kurtosis_all = pd.concat([df_kurtosis_B, df_kurtosis_S, df_kurtosis_Sch ]).reset_index(drop=True)
```

In [25]:

```
df_mean_all.to_csv('D:\\Uni\Diplomarbeit\Diplomarbeit-Daten\df_all_mean.csv')    # Exportieren des großen df
df_skew_all.to_csv('D:\\Uni\Diplomarbeit\Diplomarbeit-Daten\df_all_skew.csv')
df_kurtosis_all.to_csv('D:\\Uni\Diplomarbeit\Diplomarbeit-Daten\df_all_kurtosis.csv')
```

Anhang B: Python-Codes-Teilmengen

Teilmenge-Mittelwert

In [1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import os
import warnings
warnings.filterwarnings('ignore')

from pandas import read_csv

import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df_all_mittelwert = pd.read_csv("D:\\Uni\\Diplomarbeit\\Diplomarbeit-Daten\\df_all_mean.csv")
                            # Auslesen der csv-Datei-Teilmenge-Mittelwert

del df_all_mittelwert['Unnamed: 0']  # Entfernen der unbenannten Spalten in Dataframe
df_all_mittelwert.round(3)           # Darstellung der Teilmenge-Mittelwert(nur 3 stellen nach komma)
```

Out[2]:

	1	2	3	4	5	6	7	8	9	10	11	12	Events
0	-0.002	-1.513	-0.344	0.492	-0.754	-1.061	-0.719	-0.359	-0.647	-1.777	-0.851	0.311	B
1	-0.632	-2.250	-0.749	-0.260	-0.331	-0.966	-0.881	-0.649	-0.448	-0.908	-0.929	-1.238	B
2	-0.740	-1.941	2.185	-0.207	-0.187	-1.495	-0.812	-0.700	-0.749	-0.300	-0.269	-1.453	B
3	-0.547	-1.377	1.439	-0.154	-0.511	-1.193	-0.613	-1.361	-0.557	-1.166	-0.197	-0.238	B
4	-0.835	0.298	-0.681	-0.679	-0.351	-0.163	-0.491	-0.717	-0.918	-0.121	-0.750	-0.858	B
...	...	...	...	...	...	...	...	...	...	...	...	...	...
151	-0.444	0.259	-0.321	-1.361	0.603	-1.034	-0.301	-1.185	0.066	0.153	-0.605	-0.280	Sch
152	-0.492	-1.362	0.564	-1.861	0.254	-0.802	-0.872	-1.466	-0.170	-0.136	-0.582	-0.312	Sch
153	-0.506	-0.925	1.539	-1.571	-0.053	-1.243	-0.511	-0.764	-0.171	-0.119	-0.879	-0.395	Sch
154	-0.398	0.722	-0.281	-0.272	0.363	1.114	-0.510	-0.702	-0.088	-0.120	-0.677	-0.394	Sch
155	-0.419	-0.345	0.880	-1.580	0.463	-2.324	-0.419	-0.751	-0.106	-0.184	-0.889	-0.410	Sch

156 rows × 13 columns

Scaling

In [5]:
```
# feature_cols = ['1','2','3','4','5','6','7','8','9','10','11','12']
X = df_all_mittelwert[['1','2','3','4','5','6','7','8','9','10','11','12']] # Features
y = df_all_mittelwert['Events'] # Target variable

# X=df_all.drop('Events', axis=1)
# y =df_all['Events']
```

In [6]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state = 21)
            #Trennung von Trainings- und Testdaten, % 80 train, %20 test
            # random_state (21) ist zur besseren Verteilung von  Testset

X_train.shape, X_test.shape
```

In [7]:
```
# 124 Trainingsdaten und 32 Testdaten
```

Out[7]: ((124, 12), (32, 12))

In [8]:
```
# X_train.value_counts()
```

In [9]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

# sc_X = MinMaxScaler() # Skalierung des Trainings- und Testsets
sc_X = StandardScaler()

X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

SVM-Algorithmus

In [10]:
```
from sklearn.pipeline import Pipeline
from sklearn.multiclass import OneVsRestClassifier
from sklearn.multiclass import OneVsOneClassifier

from sklearn.svm import LinearSVC
from sklearn.datasets import make_classification

from sklearn import svm
from sklearn.svm import SVC

from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Linear Kernel

In [11]:
```
linear_svc = svm.SVC(kernel='linear',C = 100,gamma = 100)
linear_svc.fit(X_train,y_train)            # Fitting für X_train und y_train
y_pred_linear = linear_svc.predict(X_test) #  die vom Modell vorhergesagten Daten

print("train accuracy:",linear_svc.score(X_train,y_train)) # accuracy of the trainingset
print("test accuracy:",linear_svc.score(X_test,y_test))    # accuracy of the testset

print(classification_report(y_test, y_pred_linear))
```

```
train accuracy: 0.8225806451612904
test accuracy: 0.6875
              precision    recall  f1-score   support

           B       0.71      0.45      0.56        11
           S       1.00      0.60      0.75        10
         Sch       0.58      1.00      0.73        11

    accuracy                           0.69        32
   macro avg       0.76      0.68      0.68        32
weighted avg       0.76      0.69      0.68        32
```

RBF (Gaussian) Kernel

In [12]:
```
rbf_svc = svm.SVC(kernel='rbf',C= 10, gamma = 0.1)
rbf_svc.fit(X_train,y_train)
y_pred_rbf = rbf_svc.predict(X_test)


print("train accuracy:",rbf_svc.score(X_train,y_train))
print("test accuracy:",rbf_svc.score(X_test,y_test))

print(classification_report(y_test, y_pred_rbf))
```

```
train accuracy: 0.5645161290322581
test accuracy: 0.4375
              precision    recall  f1-score   support

           B       0.38      1.00      0.55        11
           S       1.00      0.30      0.46        10
         Sch       0.00      0.00      0.00        11

    accuracy                           0.44        32
   macro avg       0.46      0.43      0.34        32
weighted avg       0.44      0.44      0.33        32
```

Polynomial Kernel

In [13]:
```
poly_svc = svm.SVC(kernel='poly',C=10,degree = 3,gamma =1 )
poly_svc.fit(X_train,y_train)
y_pred_poly = poly_svc.predict(X_test)

print("train accuracy:",poly_svc.score(X_train,y_train))

print("test accuracy:",poly_svc.score(X_test,y_test))

print(classification_report(y_test, y_pred_poly))
```

```
train accuracy: 0.6693548387096774
test accuracy: 0.40625
              precision    recall  f1-score   support

           B       0.37      0.64      0.47        11
           S       1.00      0.50      0.67        10
         Sch       0.12      0.09      0.11        11

    accuracy                           0.41        32
   macro avg       0.50      0.41      0.41        32
weighted avg       0.48      0.41      0.40        32
```

Sigmoid Kernel

In [14]:

```
sigmoid_svc= svm.SVC(kernel='sigmoid',C=10,gamma = 10)
sigmoid_svc.fit(X_train, y_train)
y_pred_sigmoid = sigmoid_svc.predict(X_test)

print("train accuracy:",sigmoid_svc.score(X_train,y_train))

print("test accuracy:",sigmoid_svc.score(X_test,y_test))

print(classification_report(y_test, y_pred_sigmoid))
```

```
train accuracy: 0.4596774193548387
test accuracy: 0.34375
              precision    recall  f1-score   support

           B       0.38      0.55      0.44        11
           S       1.00      0.10      0.18        10
         Sch       0.27      0.36      0.31        11

    accuracy                           0.34        32
   macro avg       0.55      0.34      0.31        32
weighted avg       0.53      0.34      0.32        32
```

GridSearch / k-Fold CV

In [15]:

```
from sklearn.pipeline import Pipeline
```

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.multiclass import OneVsOneClassifier

from sklearn.svm import LinearSVC
from sklearn.datasets import make_classification

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn import svm
from sklearn.svm import SVC

from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

In [17]:

```
# Erforderliche Parameter für Gridsearch

C=[100,10,1,0.01,0.001,0.1] # Fehlergewicht
gamma=[0.01,0.001,0.0001,1,10,25,100]

kernel=["linear","rbf","poly","sigmoid"]
degree=[1,2,3,8]
decision_function_shape=["ovo","ovr"]  # 'One vs One' oder 'One vs Rest'

Rkfold=RepeatedKFold(n_splits=10,n_repeats= 50, random_state = 21)  # 50 Wiederholte 10-fach Kreuzvalidierung
Skfold=StratifiedKFold(n_splits=10, shuffle=True,random_state = 21) # stratifizierte 10-fach Kreuzvalidierung
kfold=KFold(n_splits=10,shuffle=True, random_state = 21)            # normal 10-fach Kreuzvalidierung
```

In [18]:

```
# Anwenden von Parametern auf 'Gridsearch'
svm=SVC(random_state = 21)

grid_clf=GridSearchCV(estimator=svm,cv=Skfold,refit=True,param_grid=dict(kernel=kernel,
                                                                         C=C,
                                                                         gamma=gamma,
                                                                         degree=degree,
```

```
                                                                    decision_function_shape=decision_function_shape))

grid_clf.fit(X_train,y_train)

y_pred_grid = grid_clf.predict(X_test)
```

In [19]:
```
print("best score: ", grid_clf.best_score_)
print("best param: ", grid_clf.best_params_)

#höchste Punktzahl und beste Parameter
```

```
best score:  0.808974358974359
best param:  {'C': 100, 'decision_function_shape': 'ovo', 'degree': 1, 'gamma': 100, 'kernel': 'poly'}
```

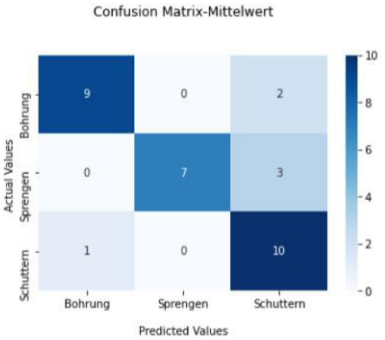
In [21]:
```
print("Accuracy score %s" %accuracy_score(y_test,y_pred_grid))
print("Classification report  \n %s" %(classification_report(y_test, y_pred_grid)))
```

```
Accuracy score 0.8125
Classification report
              precision    recall  f1-score   support

           B       0.90      0.82      0.86        11
           S       1.00      0.70      0.82        10
         Sch       0.67      0.91      0.77        11

    accuracy                           0.81        32
   macro avg       0.86      0.81      0.82        32
weighted avg       0.85      0.81      0.82        32
```

In [22]:
```
cv_results = pd.DataFrame(grid_clf.cv_results_)
best_model_results = cv_results.loc[grid_clf.best_index_]
best_model_results
```

Out[22]:
```
mean_fit_time                                                    0.046864
std_fit_time                                                     0.017112
mean_score_time                                                  0.001566
std_score_time                                                   0.004698
param_C                                                               100
param_decision_function_shape                                         ovo
param_degree                                                            1
param_gamma                                                           100
param_kernel                                                         poly
params                        {'C': 100, 'decision_function_shape': 'ovo', '...
split0_test_score                                                0.846154
split1_test_score                                                0.769231
split2_test_score                                                0.692308
split3_test_score                                                0.615385
split4_test_score                                                     1.0
split5_test_score                                                0.916667
split6_test_score                                                    0.75
split7_test_score                                                    0.75
split8_test_score                                                     1.0
split9_test_score                                                    0.75
mean_test_score                                                  0.808974
std_test_score                                                   0.122091
rank_test_score                                                         1
Name: 26, dtype: object
```

In [23]:
```
y_pred_grid = grid_clf.predict(X_test)

# print(sns.heatmap(confusion_matrix(y_test,y_pred_grid),annot=True))

# print(classification_report(y_test,y_pred_grid))#Output


y_pred_grid = confusion_matrix(y_test,y_pred_grid)
ax = sns.heatmap(y_pred_grid, annot=True, cmap='Blues')

ax.set_title('Confusion Matrix-Mittelwert\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

##
ax.xaxis.set_ticklabels(['Bohrung','Sprengen','Schuttern'])
ax.yaxis.set_ticklabels(['Bohrung','Sprengen','Schuttern'])

## Display the visualization of the Confusion Matrix.
plt.show()
```



Teilmenge-Schiefe

In [1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import os
import warnings
warnings.filterwarnings('ignore')

import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df_all_Schiefe = pd.read_csv("D:\\Uni\\Diplomarbeit\\Diplomarbeit-Daten\\df_all_skew.csv")
del df_all_Schiefe['Unnamed: 0']
df_all_Schiefe.round(3)
```

Out[2]:

	1	2	3	4	5	6	7	8	9	10	11	12	Events
0	-0.077	-0.122	-0.016	0.340	-0.002	-0.184	-0.017	0.476	0.007	-0.133	0.032	0.130	B
1	0.002	-0.072	-0.005	0.035	0.026	-0.182	0.041	0.058	0.014	-0.035	0.067	-0.413	B
2	-0.013	-0.150	-0.411	0.110	0.008	-0.285	-0.009	0.077	0.009	0.056	0.018	-0.189	B
3	-0.007	-0.351	-2.485	0.076	-0.013	-0.369	-0.023	0.029	-0.021	-0.435	0.042	-0.322	B
4	-0.022	0.116	0.173	-0.085	0.025	0.314	0.015	0.266	0.001	0.243	-0.019	0.031	B
...	...	...	...	...	...	...	...	...	...	...	...	...	...
151	-1.894	0.039	-0.007	-0.033	-0.674	-0.053	-0.001	-0.095	-0.176	-0.103	0.627	-0.345	Sch
152	-1.064	0.014	0.055	0.516	2.053	-0.028	0.297	0.289	-0.187	0.178	1.046	1.018	Sch
153	-10.985	0.028	-1.331	0.336	-3.979	-0.042	0.281	0.371	3.244	-0.055	2.814	0.829	Sch
154	-2.578	0.083	-0.376	0.098	-1.448	0.055	0.071	0.097	0.444	0.031	-3.448	0.536	Sch
155	-4.786	1.879	0.368	-0.007	-0.559	-0.081	-0.002	-0.091	0.584	-0.193	-0.488	-0.217	Sch

156 rows × 13 columns

In [ ]:
```
```

```
fig, ax = plt.subplots(1, figsize=(18, 6))
plt.title("Schiefe")
plt.xlabel("Anzahl von Daten")
plt.ylabel("Amplitude [Hz]")
plt.plot( df_all[['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']], color = "red")
ax.figure.savefig('Bohrung.jpg')
```

Scaling

In [3]:
```
X = df_all_Schiefe[['1','2','3','4','5','6','7','8','9','10','11','12']] # Features
y = df_all_Schiefe['Events'] # Target variable
```

In [4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 21)
```

In [5]:
```
X_train.shape, X_test.shape
```

Out[5]: ((124, 12), (32, 12))

In [6]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

# sc_X = MinMaxScaler()
sc_X = StandardScaler()

X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

SVM-Algorithmus

In [7]:
```
from sklearn.pipeline import Pipeline
from sklearn.multiclass import OneVsRestClassifier
from sklearn.multiclass import OneVsOneClassifier
```

```
from sklearn.svm import LinearSVC
from sklearn.datasets import make_classification

from sklearn import svm
from sklearn.svm import SVC

from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

linear Kernel

In [8]:

```
linear_svc = svm.SVC(kernel='linear',C = 100,gamma = 100)
linear_svc.fit(X_train,y_train)
y_pred_linear = linear_svc.predict(X_test)

print("train accuracy:",linear_svc.score(X_train,y_train))
print("test accuracy:",linear_svc.score(X_test,y_test))

print(classification_report(y_test, y_pred_linear))
```

```
train accuracy: 0.7580645161290323
test accuracy: 0.59375
              precision    recall  f1-score   support

           B       0.53      0.73      0.62        11
           S       0.88      0.70      0.78        10
         Sch       0.44      0.36      0.40        11

    accuracy                           0.59        32
   macro avg       0.62      0.60      0.60        32
weighted avg       0.61      0.59      0.59        32
```

RBF Kernel

In [9]:

```
rbf_svc = svm.SVC(kernel='rbf',C=10,gamma = 0.1)
```

```
rbf_svc.fit(X_train,y_train)
y_pred_rbf = rbf_svc.predict(X_test)


print("train accuracy:",rbf_svc.score(X_train,y_train))
print("test accuracy:",rbf_svc.score(X_test,y_test))

print(classification_report(y_test, y_pred_rbf))
```

```
train accuracy: 0.8709677419354839
test accuracy: 0.625
              precision    recall  f1-score   support

           B       0.77      0.91      0.83        11
           S       0.56      0.90      0.69        10
         Sch       0.33      0.09      0.14        11

    accuracy                           0.62        32
   macro avg       0.56      0.63      0.56        32
weighted avg       0.55      0.62      0.55        32
```

Polynomial Kernel

In [10]:

```
poly_svc = svm.SVC(kernel='poly',C=10,degree = 3,gamma =1)
poly_svc.fit(X_train,y_train)
y_pred_poly = poly_svc.predict(X_test)

print("train accuracy:",poly_svc.score(X_train,y_train))

print("test accuracy:",poly_svc.score(X_test,y_test))

print(classification_report(y_test, y_pred_poly))
```

```
train accuracy: 0.9838709677419355
test accuracy: 0.71875
              precision    recall  f1-score   support

           B       0.71      0.91      0.80        11
           S       0.71      1.00      0.83        10
         Sch       0.75      0.27      0.40        11

    accuracy                           0.72        32
```

```
   macro avg       0.73      0.73      0.68        32
weighted avg       0.73      0.72      0.67        32
```

Sigmoid Kernel

In [11]:
```
sigmoid_svc= svm.SVC(kernel='sigmoid',C=10,gamma = 10)
sigmoid_svc.fit(X_train, y_train)
y_pred_sigmoid = sigmoid_svc.predict(X_test)

print("train accuracy:",sigmoid_svc.score(X_train,y_train))

print("test accuracy:",sigmoid_svc.score(X_test,y_test))

print(classification_report(y_test, y_pred_sigmoid))
```

```
train accuracy: 0.33064516129032256
test accuracy: 0.3125
              precision    recall  f1-score   support

           B       0.60      0.55      0.57        11
           S       0.22      0.20      0.21        10
         Sch       0.15      0.18      0.17        11

    accuracy                           0.31        32
   macro avg       0.33      0.31      0.32        32
weighted avg       0.33      0.31      0.32        32
```

In [ ]:
```

```

GridSearch / k-Fold CV

In [12]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import KFold
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import cross_val_score

from numpy import mean
from numpy import std
```

In [13]:
```
C=[1000,100,10,1,0.1]
gamma=[0.01,0.1,1,10,25,100]

kernel=["linear","rbf","poly","sigmoid"]
degree=[1,2,3]
decision_function_shape=["ovo","ovr"]



Rkfold=RepeatedKFold(n_splits=10,n_repeats= 50, random_state = 21)
Skfold=StratifiedKFold(n_splits=10, shuffle=True, random_state = 21)
kfold=KFold(n_splits=10,shuffle=True, random_state = 21)
```

In [14]:
```
svm=SVC(random_state = 21)

grid_clf=GridSearchCV(estimator=svm,cv=Skfold,param_grid=dict(kernel=kernel,C=C, gamma=gamma,degree=degree, decision_function_shap
grid_clf.fit(X_train,y_train)

y_pred_grid = grid_clf.predict(X_test)
```

In [15]:
```
print("best score: ", grid_clf.best_score_)
print("best param: ", grid_clf.best_params_)
```

```
best score:  0.7410256410256411
best param:  {'C': 10, 'decision_function_shape': 'ovo', 'degree': 1, 'gamma': 1, 'kernel': 'rbf'}
```

In [16]:
```
print("Accuracy score %s" %accuracy_score(y_test, y_pred_grid))
print("Classification report  \n %s" %(classification_report(y_test, y_pred_grid)))
```

```
Accuracy score 0.6875
Classification report
              precision    recall  f1-score   support

           B       0.91      0.91      0.91        11
           S       0.56      0.90      0.69        10
         Sch       0.60      0.27      0.37        11
```

```
              accuracy                           0.69        32
             macro avg       0.69      0.69      0.66        32
          weighted avg       0.69      0.69      0.66        32
```

In [17]:

```
cv_results = pd.DataFrame(grid_clf.cv_results_)
best_model_results = cv_results.loc[grid_clf.best_index_]
best_model_results
```

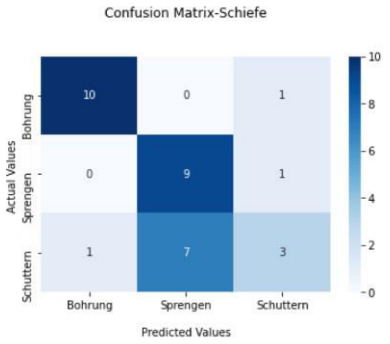
Out[17]:

```
mean_fit_time                                                    0.002394
std_fit_time                                                     0.000488
mean_score_time                                                  0.000598
std_score_time                                                   0.000488
param_C                                                                10
param_decision_function_shape                                         ovo
param_degree                                                            1
param_gamma                                                             1
param_kernel                                                          rbf
params                          {'C': 10, 'decision_function_shape': 'ovo', 'd...
split0_test_score                                                0.769231
split1_test_score                                                0.846154
split2_test_score                                                0.692308
split3_test_score                                                0.769231
split4_test_score                                                0.666667
split5_test_score                                                0.583333
split6_test_score                                                    0.75
split7_test_score                                                0.916667
split8_test_score                                                0.916667
split9_test_score                                                     0.5
mean_test_score                                                  0.741026
std_test_score                                                   0.128564
rank_test_score                                                         1
Name: 297, dtype: object
```

In [18]:

```
y_pred_grid = grid_clf.predict(X_test)

# print(sns.heatmap(confusion_matrix(y_test,y_pred_grid),annot=True))

# print(classification_report(y_test,y_pred_grid))#Output


y_pred_grid = confusion_matrix(y_test,y_pred_grid)
ax = sns.heatmap(y_pred_grid, annot=True, cmap='Blues')
```

```
ax.set_title('Confusion Matrix-Schiefe\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

##
ax.xaxis.set_ticklabels(['Bohrung','Sprengen','Schuttern'])
ax.yaxis.set_ticklabels(['Bohrung','Sprengen','Schuttern'])

## Display the visualization of the Confusion Matrix.
plt.show()
```



Teilmenge-Wölbung

In [1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import os
import warnings
warnings.filterwarnings('ignore')

import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
```

In [2]:
```
df_all_Wölbung = pd.read_csv("D:\\Uni\\Diplomarbeit\\Diplomarbeit-Daten\\df_all_kurtosis.csv")
del df_all_Wölbung['Unnamed: 0']
df_all_Wölbung.round(3)
```

Out[2]:

	1	2	3	4	5	6	7	8	9	10	11	12	Events
0	9.604	5.946	11.575	6.033	23.615	7.401	14.525	7.762	12.257	7.068	5.365	7.189	B
1	6.275	7.080	8.113	7.795	10.388	6.810	12.984	4.964	10.966	5.646	10.539	6.884	B
2	17.441	6.447	36.130	6.204	19.563	9.513	11.960	6.163	83.154	6.400	15.281	9.377	B
3	27.647	11.120	293.755	10.604	36.089	14.920	14.821	14.392	25.213	13.300	21.145	19.222	B
4	11.697	4.790	13.465	4.600	16.348	6.012	13.731	5.713	12.892	3.292	23.734	3.671	B
...	...	...	...	...	...	...	...	...	...	...	...	...	...
151	287.577	-0.816	-0.897	96.549	311.920	-0.274	5.557	205.079	1298.712	576.254	140.553	313.485	Sch
152	380.215	-0.855	6.076	362.289	688.922	-0.514	67.199	647.796	2360.826	475.475	723.963	1123.073	Sch
153	5113.500	-0.904	1718.988	997.298	3871.076	-0.625	5767.863	1915.330	3497.621	1494.807	7350.114	1751.610	Sch
154	1287.836	0.774	1414.115	919.733	1738.963	0.254	3408.907	1525.746	3334.590	794.119	2125.491	1100.484	Sch
155	2063.998	562.704	748.986	2814.452	2247.484	-0.448	1749.060	1685.599	1752.706	1470.320	2720.531	2987.975	Sch

156 rows × 13 columns

In [3]:
```
fig, ax = plt.subplots(1, figsize=(18, 6))

plt.title("Wölbung")
plt.xlabel("Anzahl von Daten")
plt.ylabel("Amplitude [Hz]")

plt.plot(df_all[['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']], color = "red")
ax.figure.savefig('Bohrung.jpg')
```

Scaling

In [6]:
```
X = df_all_Wölbung[['1','2','3','4','5','6','7','8','9','10','11','12']] # Features
y = df_all_Wölbung['Events'] # Target variable
```

In [7]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state = 21)
```

In [8]:
```
X_train.shape, X_test.shape
```

Out[8]:
```
((124, 12), (32, 12))
```

In [9]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

# sc_X = MinMaxScaler()
sc_X = StandardScaler()

X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

SVM-Algorithmus

In [10]:
```
from sklearn.pipeline import Pipeline


from sklearn.svm import LinearSVC
from sklearn.datasets import make_classification

from sklearn import svm
from sklearn.svm import SVC


from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Linear Kernel

In [11]:
```
linear_svc = svm.SVC(kernel='linear',C = 100,gamma = 100)
linear_svc.fit(X_train,y_train)
y_pred_linear = linear_svc.predict(X_test)

print("train accuracy:",linear_svc.score(X_train,y_train))
print("test accuracy:",linear_svc.score(X_test,y_test))

print(classification_report(y_test, y_pred_linear))
```

```
train accuracy: 0.9758064516129032
test accuracy: 0.90625
              precision    recall  f1-score   support

           B       1.00      1.00      1.00        11
           S       0.89      0.80      0.84        10
         Sch       0.83      0.91      0.87        11

    accuracy                           0.91        32
   macro avg       0.91      0.90      0.90        32
weighted avg       0.91      0.91      0.91        32
```

RBF(Gaussian) Kernel

In [12]:
```
rbf_svc = svm.SVC(kernel='rbf',C=10,gamma = 1)
rbf_svc.fit(X_train,y_train)
y_pred_rbf = rbf_svc.predict(X_test)


print("train accuracy:",rbf_svc.score(X_train,y_train))
print("test accuracy:",rbf_svc.score(X_test,y_test))

print(classification_report(y_test, y_pred_rbf))
```

```
train accuracy: 0.9838709677419355
test accuracy: 0.90625
              precision    recall  f1-score   support

           B       1.00      1.00      1.00        11
           S       1.00      0.70      0.82        10
         Sch       0.79      1.00      0.88        11

    accuracy                           0.91        32
   macro avg       0.93      0.90      0.90        32
weighted avg       0.93      0.91      0.90        32
```

Polynomial Kernel

In [13]:
```
poly_svc = svm.SVC(kernel='poly' ,C=10,degree = 3,gamma =1)
poly_svc.fit(X_train,y_train)
y_pred_poly = poly_svc.predict(X_test)

print("train accuracy:",poly_svc.score(X_train,y_train))

print("test accuracy:",poly_svc.score(X_test,y_test))

print(classification_report(y_test, y_pred_poly))
```

```
train accuracy: 0.9919354838709677
test accuracy: 0.84375
              precision    recall  f1-score   support
```

```
           B       1.00      1.00      1.00        11
           S       0.73      0.80      0.76        10
         Sch       0.80      0.73      0.76        11

    accuracy                           0.84        32
   macro avg       0.84      0.84      0.84        32
weighted avg       0.85      0.84      0.84        32
```

Sigmoid Kernel

In [14]:
```
sigmoid_svc= svm.SVC(kernel='sigmoid',C=10,gamma = 10)
sigmoid_svc.fit(X_train, y_train)
y_pred_sigmoid = sigmoid_svc.predict(X_test)

print("train accuracy:",sigmoid_svc.score(X_train,y_train))

print("test accuracy:",sigmoid_svc.score(X_test,y_test))

print(classification_report(y_test, y_pred_sigmoid))
```

```
train accuracy: 0.5806451612903226
test accuracy: 0.71875
              precision    recall  f1-score   support

           B       0.58      1.00      0.73        11
           S       0.80      0.40      0.53        10
         Sch       1.00      0.73      0.84        11

    accuracy                           0.72        32
   macro avg       0.79      0.71      0.70        32
weighted avg       0.79      0.72      0.71        32
```

GridSearch / k-Fold CV

In [15]:
```
from sklearn.model_selection import GridSearchCV

from sklearn.multiclass import OneVsRestClassifier
from sklearn.multiclass import OneVsOneClassifier
```

```
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import KFold
from sklearn.model_selection import RepeatedKFold

from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve, auc


from numpy import mean
from numpy import std
from pandas import Series,DataFrame
```

In [ ]:
```
```

In [16]:
```
C=[100,10,1,0.01,0.001,0.1]
gamma=[0.01,0.001,0.0001,1,10]

kernel=["linear","rbf","poly","sigmoid"]
degree=[1,2,3]
decision_function_shape=["ovo","ovr"]



Rkfold=RepeatedKFold(n_splits=10,n_repeats= 50, random_state =21)
Skfold=StratifiedKFold(n_splits=10, shuffle=True,random_state = 21)
kfold=KFold(n_splits=10,shuffle=True, random_state = 21)
```

In [ ]:
```
```

In [17]:
```
svm=SVC(random_state = 21)

grid_clf=GridSearchCV(estimator=svm,cv=Skfold,refit=True,param_grid=dict(kernel=kernel,C=C, gamma=gamma,degree=degree, decision_fu
grid_clf.fit(X_train,y_train)


y_pred_grid = grid_clf.predict(X_test)
```

In [18]:
```
print("best score: ", grid_clf.best_score_)
print("best param: ", grid_clf.best_params_)
```

```
best score:  0.9044871794871796
best param:  {'C': 100, 'decision_function_shape': 'ovo', 'degree': 1, 'gamma': 1, 'kernel': 'rbf'}
```

In [19]:
```
print("Accuracy score %s" %accuracy_score(y_test, y_pred_grid))
print("Classification report  \n %s" %(classification_report(y_test, y_pred_grid)))
```

```
Accuracy score 0.90625
Classification report
              precision    recall  f1-score   support

           B       0.92      1.00      0.96        11
           S       1.00      0.70      0.82        10
         Sch       0.85      1.00      0.92        11

    accuracy                           0.91        32
   macro avg       0.92      0.90      0.90        32
weighted avg       0.92      0.91      0.90        32
```

In [20]:
```
cv_results = pd.DataFrame(grid_clf.cv_results_)
best_model_results = cv_results.loc[grid_clf.best_index_]
best_model_results
```

Out[20]:
```
mean_fit_time                                                    0.001562
std_fit_time                                                     0.004686
mean_score_time                                                  0.001562
std_score_time                                                   0.004685
param_C                                                               100
param_decision_function_shape                                         ovo
param_degree                                                            1
param_gamma                                                             1
param_kernel                                                          rbf
params                          {'C': 100, 'decision_function_shape': 'ovo', '...
split0_test_score                                                0.923077
split1_test_score                                                0.846154
split2_test_score                                                0.923077
split3_test_score                                                0.769231
split4_test_score                                                     1.0
split5_test_score                                                     1.0
split6_test_score                                                0.916667
split7_test_score                                                0.916667
split8_test_score                                                0.916667
split9_test_score                                                0.833333
mean_test_score                                                  0.904487
std_test_score                                                   0.067928
rank_test_score                                                         1
Name: 13, dtype: object
```

In [21]:
```
y_pred_grid = grid_clf.predict(X_test)

# print(sns.heatmap(confusion_matrix(y_test,y_pred_grid),annot=True))

# print(classification_report(y_test,y_pred_grid))#Output


y_pred_grid = confusion_matrix(y_test,y_pred_grid)
ax = sns.heatmap(y_pred_grid, annot=True, cmap='Blues')

ax.set_title('Confusion Matrix-Wölbung\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

##
ax.xaxis.set_ticklabels(['Bohrung','Sprengen','Schuttern'])
ax.yaxis.set_ticklabels(['Bohrung','Sprengen','Schuttern'])

## Display the visualization of the Confusion Matrix.
plt.show()
```