Otto-Friedrich Universität Bamberg
Lehrstuhl für Statistik und Ökonometrie
Statistical Machine Learning

Support Vector Machines

Autoren:

Niklas Münz
Matrikel Nr.: 2150715

Franz Andersch
Matrikel Nr.: 2154877

Studiengang:

MSc. Survey Statistics
& Data Analysis

Sommersemester 2024

Inhaltsverzeichnis

1 Einleitung

Bei der Separierung von zwei Datenklassen scheint eine lineare Trennlinie zwischen die Klassen zu ziehen eine simple und intuitive Methode zu sein. Genau auf dieser Idee basieren *Support Vector Machines* (*SVM*). Jedoch sind Daten nicht immer perfekt trennbar oder es liegt keine lineare Trennlinie vor. Wie *SVM* funktionieren, wie sie mit solchen Situationen umgehen, wie sie unter vorgegebenen Bedingungen performen und welche Schlussfolgerungen für die praktische Anwendung gezogen werden können, soll in dieser Arbeit beleuchtet werden.

Die Grundlage für eine optimal separierende Hyperebene wurde bereits 1964 von Alexej Chervonenkis und Vladimir Vapnik gelegt und die Methode wurde anschließend von mehreren Autoren stetig erweitert (Vapnik, 2006). Seitdem ist ihre Leistung, trotz großer Fortschritte im Bereich neuronaler Netze, vor allem im Bereich binärer Klassifikationsprobleme unumstritten und sie gelten dabei als eine der meistgenutzten Methoden (Sofi & Awan, 2017). Dabei bieten *SVM* eine große Anzahl an Anwendungsmöglichkeiten, wie beispielsweise, in der Bildklassifikation, in der Bioinformatik (Krebsklassifikation) oder im Aufdecken von Kreditkartenbetrug (Cervantes et al., 2020).

In dieser Arbeit wollen wir zuerst die Funktionsweise der *SVM* näher betrachten. Dazu wird zunächst gezeigt wie die Konstruktion dieser separierenden Hypereben im Falle linear trennbarer Daten funktioniert. Daraufhin wird die Idee eines Soft Margin Classifier aufgegriffen, der sich dem Problem nicht trennbarer Daten annimmt. Zuletzt geht es um die Anwendung von Kernels, die es ermöglichen, nicht lineare Entscheidungsgrenzen herzustellen.
In Kapitel 3 werden dann die Vor- und Nachteile der Methode betrachtet. Anhand dieser Informationen werden Schlussfolgerungen zur Leistung der *SVM* gezogen. Darauf aufbauend wird im folgenden Kapitel der Aufbau unseres Experiments beschrieben. Wir generieren uns zum anschließenden Vergleich neun verschiedene Datenszenarien, welche in ihrer Dimensionalität und Komplexität der Entscheidungsgrenze variieren. Für die Evaluierung der Klassifikationsleistung werden neben den *SVM* weitere Klassifikationsmethoden herangezogen.
Im Kapitel Hypothesen wurden auf Basis von Literatur, in der bereits ähnliche Versuche durchgeführt wurden, Vermutungen aufgestellt, wie die einzelnen Klassifikationsmethoden im Vergleich abschneiden werden. Es folgen die Ergebnisse, in denen die Durchführung des Experiments beschrieben und die Leistungen anhand verschiedener Maßzahlen evaluiert werden. Darüber hinaus wird überprüft, ob sich die Hypothesen die wir zuvor aufgestellt haben bewahrheitet haben. Im letzten Abschnitt wird ein Fazit gezogen, indem die Ergebnisse diskutiert und Weiterentwicklungsmöglichkeiten für unsere Arbeit aufgearbeitet werden.

2 Funktionsweise

2.1 Hard Margin Classifier

Um das grundlegende Prinzip der *SVM* darzustellen, gehen wir zuerst von einer Datensituation aus, in der sich zwei Gruppen optimal durch eine lineare Entscheidungsgrenze trennen lassen. Das endgültige Ziel ist es eine sogenannte Hyperplane zu finden die diese Daten möglichst gut separiert und als Entscheidungsgrenze funktioniert. Die allgemeine Form einer solchen Hyperplane lautet

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n = 0 \tag{1}$$

oder in Vektorschreibweise

$$\overline{\beta} \cdot \overline{x} + \beta_0 = 0 \tag{2}$$

Die geometrische Interpretation des Vektors β und des Skalars β_0 wird in Abbildung 1 im zweidimensionalen Fall dargestellt.

Die blaue Linie soll die Hyperplane darstellen. Im zweidimensionalen handelt es sich hier um eine Linie. Der 2×1 Vektor $\overline{\beta}$ liegt immer senkrecht zur konstruierten Hyperplane. Würde man alle Vektoren, die auf der

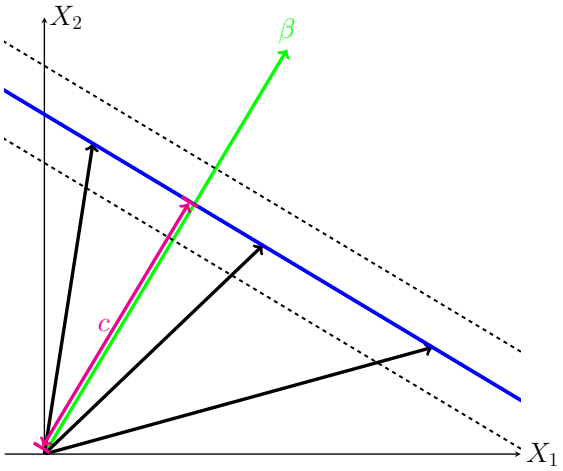

Abbildung 1: Konstruktion der Hyperebene

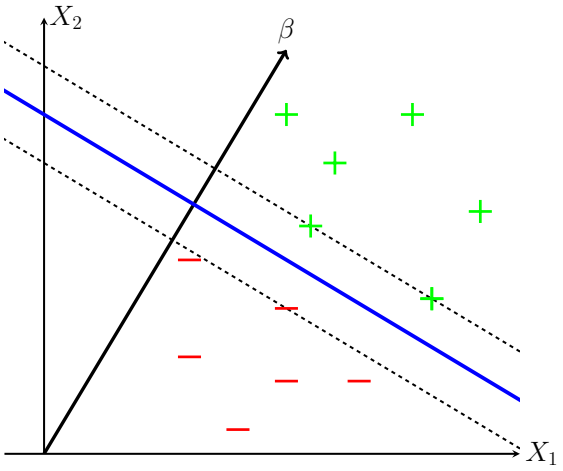

Abbildung 2: Konstruktion der Margins

Hyperplane landen, auf den normierten $\overline{\beta}$-Vektor projizieren, dann erhält man für alle diese Projektionen denselben Wert c. Also gilt für alle Punkte, die auf der Ebene liegen.

$$\frac{\overline{\beta}}{||\overline{\beta}||} \cdot \overline{x} = c \Leftrightarrow \overline{\beta} \cdot \overline{x} = c \cdot ||\overline{\beta}|| \tag{3}$$

Ersetzt man in (3) $c \cdot ||\overline{\beta}||$ mit $-\beta_0$ und zieht dies dann auf die andere Seite, kommt man wieder bei der ursprünglichen Form aus Formel (2) raus (Mavroforakis & Theodoridis, 2006).

Als Nächstes stellt sich jetzt die Frage, welches $\overline{\beta}$ und β_0 die optimale Hyperplane darstellen. Betrachtet man die Abbildung 2, dann ist zu erkennen, dass die Datenpunkte durch die blaue Linie getrennt werden. Allerdings könnte man theoretisch unendlich viele andere Hyperplanes durch Rotation oder Verschiebung konstruieren, die trotzdem die Daten in ihren Ausprägungen trennen. Um eine eindeutige Lösung zu finden, wird zunächst ein Bereich um die Hyperplane abgesteckt. In Abbildung 2 dargestellt durch die gestrichelten schwarzen Linien, welche man als Schranken bezeichnen könnte. In diesem Bereich sollen keine Datenpunkte liegen, die Schranken sollen immer parallel zur Hyperplane sein und den gleichen Abstand zu ihr haben. Außerdem dürfen keine positiven Samples unterhalb der oberen Schranke liegen und keine negativen oberhalb. Das Gegenteil gilt dementsprechend für die untere Schranke. Als Definition für die beiden Schranken wird festgelegt

$$\overline{\beta} \cdot \overline{x} + \beta_0 = 1 \tag{4}$$

für die Schranke in Richtung der grünen Datenpunkte und

$$\overline{\beta} \cdot \overline{x} + \beta_0 = -1 \tag{5}$$

für die Schranke in Richtung der roten Datenpunkte. Aus dieser Beschränkung für die Hyperplane können wir auch ableiten, dass für die positiven Samples $\overline{x}^+$ immer gilt $\overline{\beta} \cdot \overline{x}^+ + \beta_0 \geq 1$ und für negative Samples $\overline{x}^-$ immer gilt $\overline{\beta} \cdot \overline{x}^- + \beta_0 \leq -1$. Durch das Einführen einer weiteren Variable y, welche die Eigenschaft hat, dass sie den Wert 1 bei einem positiven und den Wert -1 bei einem negativen Sample annimmt, können diese zwei Beschränkungen zu Einer zusammengefasst werden (Cortes & Vapnik, 1995).

$$y_i(\overline{\beta} \cdot \overline{x}_i + \beta_0) \geq 1 \tag{6}$$

Da das Verfahren auch *Maximal Margin Classifier* gennant wird, gilt es jetzt noch eine Definition für den Margin also den Abstand zwischen den zwei Schranken zu finden, der letztendlich maximiert werden soll. Damit diese Schranken, maximal weit auseinander liegen, muss es zwangsläufig Datenpunkte geben, die genau auf den Schranken liegen. Diese Datenpunkte haben eine wichtige Rolle für die Konstruktion des

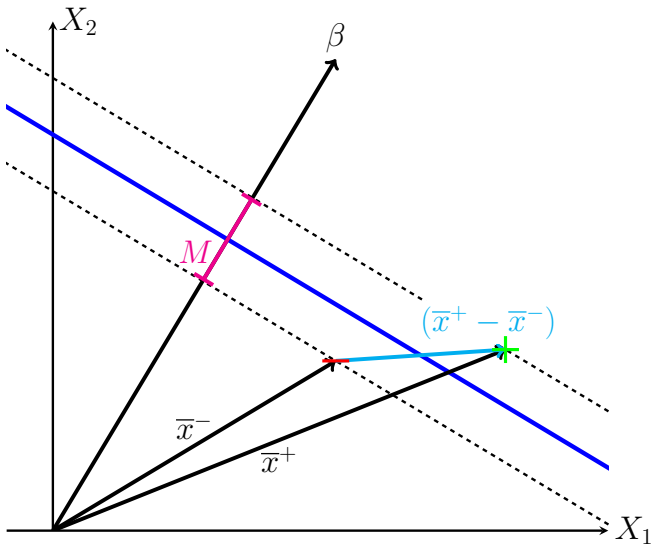

Abbildung 3: Abhängigkeit des Margins von den Support-Vektoren

Margins. Es sind ausschließlich diese Datenpunkte, die einen Einfluss auf die finalen Werte von $\overline{\beta}$ und β_0 haben werden. Sie werden *Support-Vektoren* genannt und geben den *SVM* ihren Namen.

In Abbildung 3 sind zwei solcher Support-Vektoren zu einem negativen und positiven Sample dargestellt. Der Margin kann dann dargestellt werden als eine Projektion dieser Differenz $(\overline{x}^+ - \overline{x}^+)$ auf den $\overline{\beta}$-Vektor. Damit am Ende die Länge dieses Margins M herauskommt, muss $\overline{\beta}$ noch durch seine Länge geteilt werden.

$$M = \frac{\overline{\beta}}{||\overline{\beta}||} \cdot \left(\overline{x}^+ - \overline{x}^+\right) = \frac{\overline{\beta} \cdot \overline{x}^- - \overline{\beta} \cdot \overline{x}^+}{||\overline{\beta}||} \tag{7}$$

Es ist bekannt, dass für positive Support-Vektoren gilt $\overline{\beta}\overline{x}^+ + \beta_0 = 1 \Leftrightarrow \overline{\beta}\overline{x}^+ = 1 - \beta_0$ und für negative $\overline{\beta}\overline{x}^- + \beta_0 = -1 \Leftrightarrow \overline{\beta}\overline{x}^- = -1 - \beta_0$. Setzt man dies ein in (7), erhält man als Maximierungsziel

$$M = \frac{1 - \beta_0 - (-1 - \beta_0)}{||\overline{\beta}||} = \frac{2}{||\overline{\beta}||} \tag{8}$$

Um diesen Maximierungsschritt angenehmer zu gestalten, wird an der Stelle versucht den Ausdruck $\frac{1}{2}||\overline{\beta}||^2 = \frac{1}{2}\overline{\beta}' \cdot \overline{\beta}$ zu minimieren. Was im Endeffekt ebenfalls dazu führt, dass der Ausdruck in (8) maximiert wird.

Dieses Optimierungsproblem mit der Nebenbedingung aus Formel (6) lässt sich am besten über Lagrange-Multiplier lösen

$$\mathcal{L}(\overline{\beta}, \beta_0, \overline{\alpha}) = \frac{1}{2}\overline{\beta}'\overline{\beta} - \sum \alpha_i [y_i(\overline{\beta} \cdot \overline{x_i} + \beta_0) - 1] \tag{9}$$

wobei hier $\overline{\beta}$ und β_0 minimiert und $\overline{\alpha}$ maximiert wird (Vapnik, 2006). Wird dieser Ausdruck partiell abgeleitet und gleich null gesetzt erhält man als Zwischenergebnis

$$\frac{\partial \mathcal{L}}{\partial \beta} = \beta - \sum \alpha_i y_i \overline{x_i} \stackrel{!}{=} 0 \Rightarrow \beta = \sum \alpha_i y_i \overline{x_i} \tag{10}$$

Somit zeigt sich, dass $\overline{\beta}$ als Linearkombination der Inputvektoren dargestellt werden kann. Weiterhin gilt für β_0

$$\frac{\partial \mathcal{L}}{\partial \beta_0} = \sum \alpha_i y_i \overline{x_i} \stackrel{!}{=} 0 \tag{11}$$

Setzt man dies in (9) ein erhält man einen neuen Ausdruck, denn es gilt zu minimieren

$$\mathcal{L}(\overline{\alpha}) = -\frac{1}{2}\sum\sum \alpha_i \alpha_j y_i y_j \overline{x}_i \cdot \overline{x}_j + \sum \alpha_i \tag{12}$$

Die Lösung für diesen Ausdruck erfolgt dann über sogenannte „standard non linear optimization algorithms for quadratic forms“ (Boser et al., 1992). Nachdem für $\overline{\alpha}$ gelöst wurde, kann dies in (10) eingesetzt werden, um das optimale $\overline{\beta}$ zu erhalten. Es kann gezeigt werden, dass die gelösten α_i lediglich für die Support-Vektoren Werte ungleich Null annehmen. Somit ist der Koeffizientenvektor $\overline{\beta}$ sogar eine Linearkombination von nur den Support-Vektoren (Boser et al., 1992). Die letzte unbekannte β_0 kann gelöst werden, indem man mithilfe von einem positiven/negativen Support Vektor (4)/ (5) nach β_0 löst.

Mit den gelösten Werten zur optimalen Hyperplane kann jetzt auch eine Entscheidungsregel für ungelabelte Datenvektoren $\overline{x}_u$ konstruiert werden. Bedenkt man also, wenn man einen Vektor, der nicht auf der Hyperplane liegt in (3) einsetzt, erhält man also $\frac{\overline{\beta}}{||\overline{\beta}||} \cdot \overline{x}_u = c + k$. Wenn k positiv ist, liegt der neue Datenpunkt oberhalb der Hyperplane und wird somit als positives Sample gewertet. Wenn k negativ ist, dann liegt der Datenpunkt unterhalb der Hyperplane und wird als negativ gewertet. Mit der gleichen Umformung wie weiter oben schon beschrieben kommt man zu einer Entscheidungsfunktion $f(\overline{x}_u)$ und einer daraus resultierenden Stufenfunktion $g(\overline{x}_u)$, die dann die Kategorisierung der neuen Beobachtung vornimmt.

$$g(\overline{x}_u) = \begin{cases} \text{positiv} & \text{wenn } f(\overline{x}_u) = \overline{\beta} \cdot \overline{x}_u + \beta_0 > 0 \\ \text{negativ} & \text{wenn } f(\overline{x}_u) = \overline{\beta} \cdot \overline{x}_u + \beta_0 < 0 \end{cases} \tag{13}$$

2.2 Soft Margin Classifier

Dass die Daten sich perfekt linear trennen lassen ist zwar praktisch, um die Vorgehensweise zu veranschaulichen, tritt aber in realen Situationen so gut wie nie auf. Falls sich positive und negative Samples im Raum überlappen, ist die Konstruktion einer Hyperplane wie beim *Hard Margin Classifier* unmöglich. Man müsste also entweder auf eine nicht lineare Hyperplane ausweichen oder man weicht die Vorgaben für die Konstruktion der Hyperplane etwas auf. Zweiteres ist genau das, was durch die *Soft Margin Classifier* erreicht wird. Die Vorgabe für die Konstruktion der Schranken ermöglicht es, einzelnen Datenpunkten auf der falschen Seite der Schranke, ja sogar der Entscheidungsgrenzen zu liegen. Dafür wird für die Einschränkungen eine sogenannte *Slack-Variable* ε eingeführt (James et al., 2021). Wie diese sich auswirkt, ist in Abbildung 4 aufgeführt. Setzt man diese in (6) ein, lautet die neue Nebenbedingung

$$y_i(\beta \cdot \overline{x}_i - \beta_0) > 1 - \varepsilon_i \tag{14}$$

Jetzt könnte man versuchen, diese neue Nebenbedingung einfach in das zuvor angewandte Optimierungsverfahren einzufügen. Allerdings besteht hier das Problem, dass ε einfach immer maximal groß gewählt wird und so die Bedingung immer erfüllt wird. Um das Ausmaß der Verletzung der ursprünglichen Annahmen zu begrenzen, aber trotzdem noch gewisse Abweichung zuzulassen, wird ein weiterer Parameter C, als regularisierender Parameter für ε eingeführt. Leitet man daraus zusammen mit der Restriktion $\varepsilon \geq 0$ wieder eine Lagrangefunktion her erhält man

$$\mathcal{L}(\overline{\beta}, \beta_0, \overline{\alpha}, \overline{\varepsilon}, \overline{\lambda}) = \frac{1}{2}\overline{\beta}' \cdot \overline{\beta} + C\sum_{i=1}^{n} \varepsilon_i - \underbrace{\sum \alpha_i [y_i(\overline{\beta} \cdot \overline{x_i} + \beta_0) - 1 + \varepsilon_i]}_{\text{für } y_i(\overline{\beta} \cdot \overline{x}_i - \beta_0) > 1 - \varepsilon_i} - \underbrace{\sum \lambda_i \varepsilon_i}_{\substack{\text{für} \\ \varepsilon_i \geq 0}} \tag{15}$$

Wenn dieser Ausdruck wie beim *Hard Margin Classifier* gelöst wird und die Ergebnisse eingesetzt werden erhält man wieder den Ausdruck aus (12) mit der zusätzlichen Einschränkung $0 \leq \alpha_i \leq C$ (Bennett & Campbell, 2000). Diese Min-Max-Optimierung wird dann genauso aufgelöst wie bei dem *Hard Margin Classifier* und die Entscheidungsregel ist ebenfalls gleich. Erst durch den Einsatz dieser Technik wurde von der Methode auch wirklich als *Support Vector Machine* gesprochen (Vapnik, 2006).

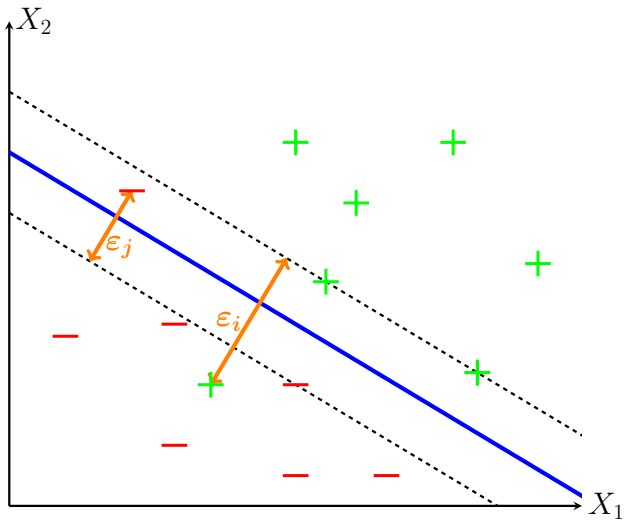

Abbildung 4: Bedeutung der Slack Variable

2.3 Der Kernel Trick

Auch wenn eine lineare Entscheidungsgrenze Vorteile in Sachen Generalisierbarkeit bietet, ist sie doch nicht für jede Datensituation geeignet.In Abbildung 5 ist es sehr gut zu erkennen, dass in diesem Fall eine lineare Grenze zwischen den Klassen keinen Sinn ergeben würde und eine elliptische Form wahrscheinlich besser geeignet wäre. Eine Lösung für dieses Problem, wäre den Merkmalsraum zu erweitern. So könnte die angenommene Formel für die lineare Hyperebene in (1) durch Polynome der Merkmale X_i oder durch Interaktionsterme erweitert werden. Dies führt dazu, dass die Entscheidungsgrenze in diesem vergrößerten Merkmalsraum immer noch linear ist, aber die Trennung möglich ist (siehe Abbildung 6). Transformiert man diese dann wieder in den ursprünglichen Merkmalsraum ist die Entscheidungsgrenze dann nicht mehr linear. Allerdings führt diese Herangehensweise zu einem starken Anstieg des Rechenaufwands, da die Möglichkeiten der Merkmalserweiterung endlos sind (James et al., 2021).

Die Lösung für das Problem sind sogenannte Kernel Funktionen. Betrachtet man die Entscheidungsfunktion (13) und setzt für $\overline{\beta}$ die Gleichung aus (10) erhält man

$$f(x_u) = \sum \alpha_i y_i x_i \cdot x_u + \beta_0 \tag{16}$$

Es zeigt sich also, dass sich die Entscheidungsfunktion im Wesentlichen aus einer Linearkombination von Punktprodukten aus dem Vektor x_u mit allen Trainingsvektoren x_i ergibt. Dieses Punktprodukt kann als Ähnlichkeitsmaß zwischen dem neuen Datenpunkt und dem jeweiligen Trainingsdatenpunkt interpretiert werden. Es ist nun möglich diese Produkte durch eine Funktion zu ersetzen, welche die Ähnlichkeiten von Datenpunkten anders bewertet. Diese sogenannte *Kernel Funktion* $K(x_i, x_j)$ ermöglicht es eine flexiblere Entscheidungsgrenze zu implementieren. Der Vorteil ist dabei, dass die Kernel Funktion nur auf alle Punktprodukte angewendet wird und es dabei nicht nötig ist den Merkmalsraum zu erweitern, was wiederum Rechenzeit spart (James et al., 2021). Die Entscheidungsfunktion wird dann mithilfe dieser *Kernel Funktionen* berechnet:

$$f(x_u) = \sum \alpha_i y_i K(x_i, x_u) + \beta_0 \tag{17}$$

Es gibt eine ganze Reihe an *Kernel Funktionen*, die bei *SVM* Anwendung finden. Die Grundlage ist der lineare Kernel, wobei dieser lediglich das Punktprodukt beschreibt, also praktisch genau das macht, was bei einer linearen Entscheidungsgrenze gemacht wird. Des Weiteren gibt es den *Polynomial Kernel*. Dieser hat die Form

$$K(x_i, x_u) = (1 + x_i \cdot x_u)^d \tag{18}$$

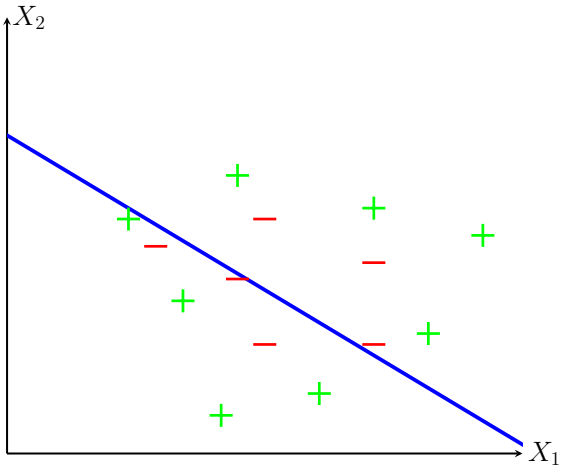

Abbildung 5: nicht linear getrennte Daten

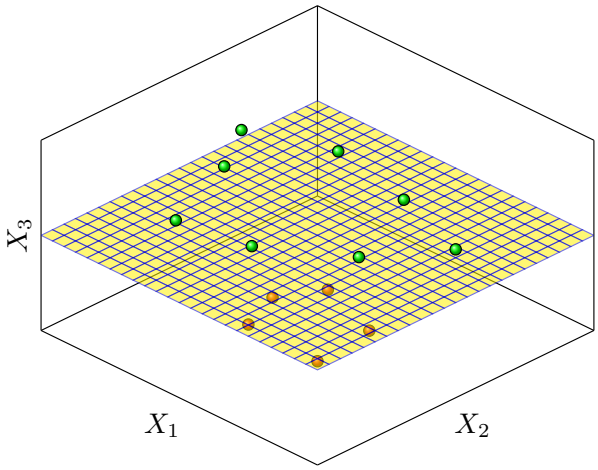

Abbildung 6: Feature Erweiterung

Die Verwendung von diesem Kernel führt dazu, dass die Entscheidungsgrenze sich ähnlich Verhält, wie als würde man zu Beginn eine Merkmalserweiterung mit Polynomen vom Grad d durchführen (siehe Abbildung 7. Eine weitere Kernel Funktion ist der *Radial Basis Function Kernel* (RBF) mit der Form

$$K(x_i, x_u) = \exp\left(-\gamma||x_i - x_u||^2\right) \tag{19}$$

Für diesen Kernel wird die quadrierte euklidische Distanz als Ähnlichkeitsmaß verwendet, was dazu führt, dass diejenigen x_i die näher an x_u liegen, in der Entscheidungsfunktion einen größeren Einfluss haben. Der Parameter γ legt dann fest, wie stark der Einfluss der Distanz sein soll. Die Projektion die der Kernel hier macht ist eine, in einen unendlich großen Merkmalsraum (James et al., 2021). Daher könnte man selbst durch vorheriges Erweitern des Merkmalsraums nicht das Ergebnis eines RBF Kernel replizieren, daher führt er auch zu einer sehr flexiblen Entscheidungsgrenze (siehe Abbildung 8).

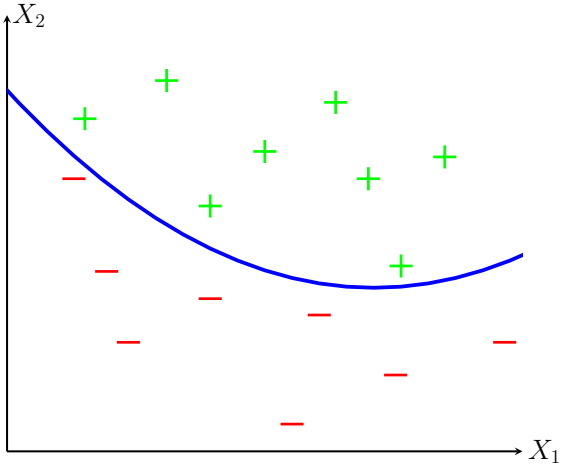

Abbildung 7: Mögliche Entscheidungsgrenze für polynomial Kernel

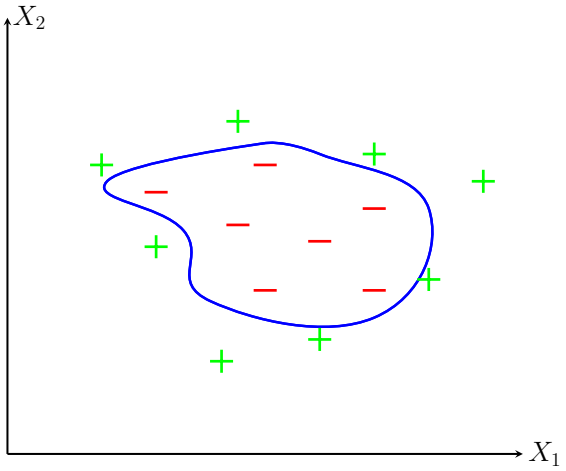

Abbildung 8: Mögliche Entscheidungsgrenze für RBF Kernel

Es gibt noch einen Reihe weitere Kernel, die auf unterschiedlichen Ähnlichkeitsmaßen beruhen, aber eher seltener oder nur in speziellen Zusammenhänge angewendet werden. Wichtig ist anzumerken, dass die Verwendung von Kernels zwar die Flexibilität der Entscheidungsgrenze erhöht, damit aber auch die Gefahr

von *Overfitting* einhergeht. Zusätzlich werden mit den Kernels auch neue Hyperparameter wie d oder γ eingeführt, die bei der Modellauswahl ebenfalls beachtet werden müssen.

3 Vor- und Nachteile der Methode

Support Vector Machines haben ein hohes Ansehen unter den Machine Learning Algorithmen, da sie einige Vorteile mit sich bringen. Aufgrund der Idee einer Soft Margin und des "Kernel Tricks" ist die Methode sehr flexibel und kann für spezielle Anwendungsbereiche angepasst werden (Bennett & Campbell, 2000). Dazu sind die Ergebnisse stabil und reproduzierbar, was sie von anderen Methoden wie beispielsweise Neural Networks abhebt. Auch die Anwendung ist vergleichsweise einfach, da es eine überschaubare Anzahl an Parametern gibt (wie beispielsweise bei der *SVM* mit radialem Kern nur der gamma- und cost-Parameter festzulegen ist).
Durch die Möglichkeit der Nutzung verschiedener Kerne sind *SVM* überaus vielseitig. Die Auswahl des Kerns ermöglicht es äußerst flexible Entscheidungsgrenzen zu formen (Kuhn & Johnson, 2013). Dadurch können *SVM* an verschiedene Datensituationen angepasst werden.
Ein weiterer Vorteil ist, dass die Methode weitgehend robust gegenüber overfitting ist (Kuhn & Johnson, 2013). Dafür verantwortlich ist der Cost-Parameter, anhand dessen der Fit an die Daten kontrolliert werden kann. Jedoch birgt dies auch Probleme (Erläuterungen im folgenden Abschnitt).
Diese Vorteile resultieren in einer allgemein häufigen Nutzung von *SVM* in der Wissenschaft, wobei sie schon oft bewiesen haben, dass sie für verschiedenste Aufgaben gut funktionieren (Kuhn & Johnson, 2013).

Trotz der vielfachen Nutzung von *SVM*, bringen sie auch Nachteile mit sich. Das wohl größte Problem liegt in der Modellselektion (Bennett & Campbell, 2000). Wie bereits im vorherigen Abschnitt erwähnt, ist die Auswahl der Parameter von hoher Bedeutung bei der Performance und dem Fit an die Daten. So kontrollieren die Kernel-spezifischen Parameter und der Cost-Parameter einerseits die Komplexität und andererseits den Fit an die Daten (Kuhn & Johnson, 2013). Dabei kann die Wahl der Parameter sowohl zu einem Underfit als auch zu einem Overfit führen. Jedoch haben nicht nur die Parameter einen Einfluss auf die Performance sondern bereits die Wahl des Kernels kann entscheidend sein (Burges, 1998). Je nach Datensituation können SVM mit verschiedenen Kernels äußerst unterschiedliche Ergebnisse liefern. Dies zeigt die Sensibilität der Methode gegenüber der Wahl des Kerns und der Parameterabstimmung.
Ein weiterer Nachteil ist, dass die Methode weniger intuitiv und aufwendiger anzuwenden ist als andere Algorithmen (Bennett & Campbell, 2000). So ist es zum Beispiel schwer Informationen aus Support Vektoren zu ziehen und es gibt keine Koeffizienten die interpretiert werden können.
Zuletzt ist zu erwähnen, dass die Methode bei einer hohen Anzahl an Beobachtungen besonders rechenintensiv ist. So konnte beispielsweise gezeigt werden, dass insbesondere die *SVM* mit polynomialem und radialem Kern eine hohe Rechenzeit aufweisen (Scholz & Wimmer, 2021). Dabei konnten andere Methoden wie die *logistische Regression* oder *k-nearest Neighbour* deutlich besser abschneiden. Dies liegt daran, dass die Lösung des *SVM*-Optimierungsproblems, die Behebung eines quadratischen Programmierungsproblems erfordert. Da die Anzahl der zu optimierenden Parameter mit der Anzahl der Daten quadratisch zunimmt, führt dies zu einer hohen Rechenkomplexität (Kecman, 2005).

4 Daten

Das Ziel dieser Arbeit ist es, in verschiedenen Datensituationen die Performance von *SVM* Algorithmen für die binäre Klassifikation zu evaluieren. Dafür wollen wir eine Reihe von Datensätzen mit verschiedenen Charakteristiken synthetisch erstellen und entsprechend als Trainingsdaten verwenden. Die Datensätze unterscheiden sich in zwei zentralen Eigenschaften. Das erste sind die Dimensionen. Es soll unterschieden werden in drei Kategorien. Die erste ist, dass es deutlich mehr Beobachtungen als Variablen gibt, also $n \gg p$. Ein solches Datenszenario kann z.B. im Kontext des Zensus auftreten, in dem eine große Anzahl an Personen befragt wird, aber die Vorgabe besteht, dass die Bürger nicht zu stark belastet werden sollen, weshalb nur einige wenige Kernfragen gestellt werden. In diesem Fall wurde für $n = 1000$ Beobachtungen und $p = 10$

Variablen generiert. Das zweite Szenario stellt Datensätze vor die etwa gleich viele Beobachtung wie Variablen haben $n \approx p$. Ein solches Szenario kann in vielen Kontexten auftreten. Für dieses Szenario sind die Dimensionen $n = p = 50$. Das letzte Szenario behandelt dann entsprechend den Fall $n \ll p$. Dies tritt oft im Kontext von Datenerhebungen im medizinischen Bereich auf, da sehr viele Erhebungen zu kostspielig wären. Auch im Bereich des Natural Language Processing sind solche Datensätze häufiger anzutreffen (Scholz & Wimmer, 2021). Die Werte die dafür angenommen wurden sind $p = 200$ und $n = 50$.

Die zweite Charakteristik ist der Datengenerierende Prozess. Da in dieser Arbeit *SVM* im Vordergrund stehen und wir hier vor allem zeigen wollen, wie *SVM* funktionieren, wird der DGP so aufgebaut, dass er der grundlegenden Idee der *SVM* am ehesten entspricht. Die Vorgehensweise ist, im ersten Schritt eine Hyperplane, im p-Dimensionalen Raum, in einer bestimmte Form zu erstellen und anschließend auf jeweils einer Seite dieser Hyperplane $n/2$ zufällige Punkte zu samplen, die die jeweilige Ausprägung in der Zielvariable repräsentieren. Wir haben also für die Zielvariable eine 50/50 Verteilung für die binären Ausprägungen angenommen.

Insgesamt gibt es auch hier wieder 3 Kategorien. Die erste sind linear getrennte Daten. Dafür wird eine lineare Hyperplane der Form

$$\beta_1 X_1 + \beta_2 X_2 + ... + \beta_{1-p} = \beta_p X_p$$

mit zufälligen Koeffizienten erzeugt. Diese Daten sollen also den Annahmen entsprechend, die für *SVM* mit linearem Kernel gelten. Nachem für eine Beobachtung j zufällig ein Punkt auf der Ebene gesampelt wurde, wurde dieser anschließend verschoben. Die Verschiebung erfolgte über eine Skalierung des normierten Normalenvektors $k\left(\frac{\overline{\beta}}{||\beta||}\right)$. k ist dabei eine normalverteilte Zufallszahl mit Mittelwert μ_k und Varianz σ_k^2. Dieser Prozess wird $n/2$ mal wiederholt für die eine Ausprägung der Zielvariable ($y_i = 1$) und dementsprechend $n/2$ mal für die andere Ausprägung ($y_i = -1$), dann aber mit $-\mu_k$ als Mittelwert für k.
In der zweiten Situation hat die Hyperplane eine quadratische Form:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_2 + \beta_4 X_2^2 + ... + \beta_{2p-2} X_{p-1} + \beta_{2p-1} X_{p-1}^2 = \beta_{2p} X_p$$

diese Form der Trennung stellt also eine Merkmalserweiterung um quadratische Terme dar und funktioniert damit ähnlich wie eine *SVM* mit polynomialen Kernel mit $d = 2$. Die Verschiebung erfolgte hier nur durch Skalierung der Werte von X_p ebenfalls mit $k \sim \mathcal{N}(\mu_k, \sigma_k^2)$ für die eine Ausprägung und $k \sim \mathcal{N}(-\mu_k, \sigma^2)$ für die andere Ausprägung.
Der letzte DGP geht von einer noch komplexeren Entscheidungsgrenzen aus. Es wird hier ein Hypershpäre im p dimensionalen Raum erstellt und einmal innerhalb und einmal außerhalb dieser gesampelt. Dafür wurde für eine Beobachtung j, $p-1$ Winkel θ zufällig erstellt, ein Radius r festgelegt und anschließend die einzelnen Werte $X_{1,j}, X_{2,j}, ..., X_{p,j}$ berechnet. Die Berechnung erfolgt dabei über die Definition von sphärischen Koordinaten:

$$\begin{aligned}
X_{1,j} &= r\cos(\theta_1) \\
X_{2,j} &= r\sin(\theta_1)\cos(\theta_2) \\
X_{3,j} &= r\sin(\theta_1)\sin(\theta_2)\cos(\theta_3) \\
&\vdots \\
X_{p-1,j} &= r\sin(\theta_1)\ldots\sin(\theta_{p-2})\cos(\theta_{p-1}) \\
X_{p,j} &= r\sin(\theta_1)\ldots\sin(\theta_{p-2})\sin(\theta_{p-1})
\end{aligned}$$

Dieser Vorgang wird dann $n/2$ mal, wiederholt und anschließend erfolgt die Verschiebung durch eine Skalierung des jeweiligen normalisierten Datenvektors $k\left(\frac{\overline{x}}{||x||}\right)$. k ist auch hier wieder eine Zufallsvariable mit μ_k für die eine Ausprägung der Zielvariable und $-\mu_k$ für die andere Ausprägung. Die Streuung bleibt bei beiden bei einem konstanten Wert σ_k^2. Je nachdem wie μ_k und σ_r^2 gewählt werden kann die Trennbarkeit der Daten angepasst werden. Für alle Szenarien wurde ein Wert für μ_k und σ_k^2 festgelegt, der für eine moderate Überschneidung der zwei Klassen sorgt. Allerdings musste der Abstand in den höherdimensionalen Szenarien

etwas erhöht werden, da sonst die Performance für alle Classifier sehr schlecht und damit nicht vergleichbar war.

Es ergeben sich daher 9 unterschiedliche Datensituationen, welche in ihren Dimensionen und Komplexität der Entscheidungsgrenze variieren. Die Kürzel für die Situationen sind in Tabelle 1 abgetragen

	linear	polynomial	radial
$p \ll n$	S1	S2	S3
$p \approx n$	S4	S5	S6
$p \gg n$	S7	S8	S9

Tabelle 1: Datenszenarien

Zusätzlich soll nicht nur ein Vergleich zwischen der Performance der *SVM* mit verschiedenen Kernels gemacht werden, sondern auch die Unterschiede in der Klassifikationsgüte der *SVM* zu anderen gängigen Klassifikationsmethoden gezeigt werden. Dafür werden *regularized Logistic Regression* und *k-nearest Neighbour* als Vergleichsalgorithmen hinzugezogen.

5 Hypothesen

Im Folgenden werden Studien hinzugezogen, um eine Einschätzung der Performance in den verschiedenen Szenarien vorzunehmen und Hypothesen abzuleiten. Vorab ist zu erwähnen, dass die Evaluation von Klassifikationsmethoden anhand synthetischer Datensätze in der Literatur begrenzt ist. Da für diese Arbeit die Form der Entscheidungsgrenze entscheidend ist, werden dennoch ausschließlich Arbeiten mit synthetischen Datensätzen zu Rate gezogen.
Aufgrund dessen, dass der datengenerierende Prozess hier so ausgearbeitet wurde, dass er mit den Annahmen der *SVM* arbeitet, erwarten wir zuerst einmal eine durchschnittlich bessere Performance der *SVM-Classifier* im Vergleich zu den anderen Methoden.

H1: Die *SVM-Classifier* performen über alle Datensituationen im Durchschnitt besser als die anderen Classifier.

Des Weiteren wurden in den einzelnen Kategorien des datengenerierenden Prozesses die Entscheidungsgrenzen speziell auf verschiedene Kernels zugeschnitten. Daher sollten *SVM* mit linearem Kernel im Setting mit linearer Entscheidungsgrenze mindestens so gut oder besser als die restlichen Classifier performen. Gleiches gilt für *SVM* mit polynomialen Kernel im Setting mit einer quadratischen Entscheidungsgrenze und radiale Kernel bei einer Hypershäre als Entscheidungsgrenze.

H2: Die *SVM-Classifier* mit dem Kernel, der für den jeweiligen DGP zugeschnitten ist sollten mindestens genauso gut oder besser performen als die restlichen Classifier.

Es konnte weiterhin gezeigt werden, dass in einem Szenario, indem erheblich mehr Beobachtungen als Dimensionen und eine lineare Entscheidungsgrenze vorliegen (S1), deutliche Unterschiede zwischen *SVM*, *K-NN* und *logistischer Regression* bei der Diskriminationsfähigkeit auftreten (Entezari-Maleki et al., 2009). *K-NN* und *SVM* mit linearem Kernel zeigen AUC-Werte nahe 1 auf, was für eine nahezu perfekte Differenzierung der Klassen spricht. Die *logistische Regression* hingegen hat einen Wert knapp über 0.5, was nur etwas besser als eine Zufallsauswahl ist. Darüber hinaus ist festzustellen, dass die Unterschiede deutlicher werden, je höher die Anzahl an Beobachtungen ist.
Für den Fall einer radialen Entscheidungsgrenze (S3) sind die Ergebnisse ähnlich. So erreicht in diesem Beispiel eine *SVM* mit radialem Kernel im Vergleich zu einer *logistischen Regression* eine um 34% höhere Genauigkeit (Fávero et al., 2022). Basierend auf den Ergebnissen der genannten Studien kann folgende Schlussfolgerung gezogen werden.

H3: In niedrigdimensionalen Szenarien performen *K-NN* und *SVM* besser als eine *logistische Regression.*

Die Szenarien S4 bis S6 finden in der Literatur kaum Beachtung, weshalb hier keine Studien herangezogen werden können. Liegt jedoch ein Szenario vor, indem die Anzahl der Dimensionen erheblich größer ist, als die Anzahl der Beobachtungen, mit einer linearen Entscheidungsgrenze (S7), sind die Ergebnisse differenziert zu betrachten. So schneidet die *SVM* mit polynomialem Kernel am besten unter den genannten Algorithmen ab, jedoch die *SVM* mit linearem Kernel am schlechtesten (als Kriterium wurde die mittlere Performance über 100 Datensätze evaluiert) (Scholz & Wimmer, 2021). Während *K-NN* auch in diesem Szenario eine gute Performance hat, schneiden *logistische Regression* und *SVM* mit radialem Kern mittelmäßig ab. Hierbei ist wichtig zu erwähnen, dass in der Studie keine Ergebnisse über die genaue Performance präsentiert wurden, sondern lediglich die Ränge der 25 behandelten Klassifikationsmethoden. Somit können nur eingeschränkte Schlussfolgerungen gezogen werden.

In hochdimensionalen Szenarien zeigt vermutlich die *SVM* mit polynomialem oder radialem Kernel eine gute Performance, unabhängig von der Form der Entscheidungsgrenze, während die linearen *SVM* voraussichtlich weniger gut abschneiden wird. Es scheint so, dass auch *K-NN* und *logistische Regression* in hochdimensionalen Szenarien zumindest mittelmäßig abschneiden. Hier ist jedoch zu beachten, dass nur eine lineare Entscheidungsgrenze betrachtet wurde und in den Szenarien S8 und S9 andere Ergebnisse möglich sind. Es ist aber auch bekannt, dass gerade die *K-NN* Methode in hochdimensionalen Szenarien schlechter performt (James et al., 2021), während *SVM* für solche Fälle sehr oft Verwendung finden (Moguerza & Muñoz, 2006). Wir schließen Final daraus:

H4: In hochdimensionalen Settings performen v.a. *SVM* mit radialen und polynomialen Kernel besser als die anderen Klassifikationsmethoden

6 Ergebnisse

6.1 Vorgehensweise bei der Analyse

Bevor die Ergebnisse erläutert werden, wird kurz auf die Vorgehensweise bei der Analyse eingegangen. In die Analyse werden fünf Modelle einbezogen: *SVM* mit linearem, polynomialem und radialem Kernel, sowie *regularisierte logistische Regression* und *k-nearest neighbours*.
Vor dem erstellen der Modelle wird ein Tuning der Hyperparameter je Modell durchgeführt. Dafür wird die Bayesian Optimization Methode genutzt, welche ein iterativer Algorithmus ist. Hierbei werden die nächsten Evaluierungspunkte basierend auf zuvor beobachteten Ergebnissen bestimmt (Yang & Shami, 2020). Der Algorithmus basiert auf zwei Hauptkomponenten: einem Surrogatmodell und einer Akqusitionsfunktion. Das Surrogatmodell, wofür hier ein Gaussian Process genutzt wird, passt die bisher beobachteten Punkte an die Zielfunktion an. Die Akquisitionsfunktion wählt dann die nächsten Punkte aus, wobei ein Gleichgewicht zwischen der Erkundung neuer Bereiche und der Nutzung vielversprechender Regionen angestrebt wird. Dafür wird hier der Ansatz des Upper-Confidence-Bound genutzt, welcher obere Konfidenzgrenzen nutzt um den Verlust gegenüber der besten möglichen Entscheidung, während der Optimierung zu minimieren (Snoek et al., 2012).

$$a_{UCB}(\mathbf{x};\ \{\mathbf{x}_n, y_n\}, \theta) = \mu(\mathbf{x};\ \{\mathbf{x}_n, y_n\}, \theta) + k\sigma(\mathbf{x};\ \{\mathbf{x}_n, y_n\}, \theta) \tag{20}$$

Die Bayesian Optimization wird genutzt, da sie eine schnelle Konvergenz für stetige Hyperparameter aufweist (Yang & Shami, 2020). Als Evaluierungskriterium wird die Genauigkeit der Modelle, welche durch den Anteil der korrekt klassifizierten Beobachtungen wiedergegeben wird, verwendet. Zur Bestimmung dieser werden die erzeugten Testdaten herangezogen.

Basierend auf den Ergebnissen des Tuning werden die oben genannten Modelle erstellt. Daraufhin werden die Genauigkeit, die Receiver Operating Characteristic Kurve (ROC-Kurve) bzw. der Area Under The Curve Wert (AUC-Wert) und der F1-Score für jedes Modell bestimmt.
Die ROC-Kurve ist eine grafische Darstellung der Leistungsfähigkeit eines Klassifikationsmodells, wobei die Sensitivität auf der y-Achse von 0 bis 1 gegen die Spezifität auf der x-Achse von 1 bis 0 abgetragen wird (Fawcett, 2006). Sensitivität und Spezifität ergeben sich aus:

$$Sensitivität = \frac{korrekt\ Positiv}{korrekt\ Positiv\ +\ falsch\ Negativ} \tag{21}$$

$$Spezifität = \frac{korrekt\ Negativ}{falsch\ Positiv\ +\ korrekt\ Negativ} \tag{22}$$

Positiv ist in diesem Fall gleichbedeutend mit Klasse 1 und Negativ mit Klasse 2. Die ROC-Kurve zeigt dann den Zusammenhang zwischen dem Nutzen (korrekt Positive) und den Kosten (falsch Positive) auf. Eine ideale Kurve läuft nah am linken, oberen Rand der Grafik, da hier bereits bei sehr hoher Spezifität (hohe Anzahl korrekt Negative) eine hohe Sensitivität (hohe Anzahl korrekt Positive) erreicht wird. Der AUC-Wert bezieht sich auf die Fläche unterhalb der Kurve und liegt somit im Intervall [0,1], wobei ein Wert von 1 für eine perfekte Klassifikation spricht, während ein Wert von 0.5 gleichbedeutend mit einer rein zufälligen Zuordnung der Klassen spricht.
Für den F1-Score ist außerdem die Präzision von Bedeutung, die sich wie folgt berechnet(Fawcett, 2006):

$$Präzision = \frac{korrekt\ Positiv}{korrekt\ Positiv\ +\ falsch\ Positiv} \tag{23}$$

Der F1-Score beschreibt das harmonische Mittel zwischen Präzision und Sensitivität und drückt folglich die Fähigkeit des Modells, gleichzeitig falsch Positive und falsch Negative zu minimieren, aus.

$$F1\text{-}Score = \frac{2}{1/Präzision\ +\ 1/Sensitivität} \tag{24}$$

Zuletzt wird innerhalb jeder Datensituation, für jeden Algorithmus ein Rang vergeben, der sich aus der Summe der drei Auswertungskriterien Genauigkeit, AUC-Wert sowie F1-Score ergibt. Dabei steht Rang 1 für den am besten abschneidenden Algorithmus in der jeweiligen Datensituation.

6.2 Darstellung der Ergebnisse

Tabelle 2 zeigt die Leistungsfähigkeit der fünf Klassifikationsalgorithmen über die neun verschiedenen Datensituationen anhand drei verschiedener Evaluationskriterien, sowie den Rang in der jeweiligen Datensituation. Betrachtet man den durchschnittlichen Rang über alle Szenarien hinweg, dann kann **H1** zumindest in Teilen bestätigt werden. Mit einem durchschnittl. Rang von 2.33 und 2.55 belegen jeweils *SVM-P* und *SVM-R* die vordersten Plätze. Allerdings muss auch gesagt, dass *SVM-L* nach den beiden anderen Klassifikationsmethoden den letzten Platz belegt. In den Datensituationen mit linearer Form der Entscheidungsgrenze performt insbesondere *logR* gut, da sie in allen drei Szenarien über alle Kriterien hinweg einen Wert von mindestens 0.72 aufweist und jeweils mindestens Rang 2 belegt. Im Falle von $p \ll n$ und $p \approx n$ zeigen auch *SVM-L*, *SVM-P* und *SVM-R* eine gute Leistung mit Werten nahe 0.9 bzw. 0.8. In dem hochdimensionalen Setting $p \gg n$ schneiden alle Drei, jedoch insbesondere *SVM-R* schlechter ab, welche einen AUC-Wert von 0.5 aufweist. *K-NN* belegt im linearen Kontext für $p \ll n$ und $p \approx n$ jeweils den fünften Rang, wobei der Unterschied im niedrigdimensionalen Raum am deutlichsten ist (0.39 schlechtere Genauigkeit als der nächst schlechteste Algorithmus). Im hochdimensionalen Raum kann *K-NN* allerdings als bester Algorithmus mit Werten über 0.8 überzeugen.
Die ROC-Kurven werden nur in den Datensituationen mit $p \ll n$ genutzt, da die grafische Darstellung in Szenarien mit kleinem n aufgrund der geringen Anzahl der Datenpunkte nicht sinnvoll erscheint. Des Weiteren ist es wichtig zu erwähnen, das für die Berechnung der ROC-Kurven die Wahrscheinlichkeit, dass eine Beobachtung zu einer bestimmten Klasse gehört, benötigt wird. Da *SVM* anders als beispielsweise *logistische Regression*, nicht direkt eine Wahrscheinlichkeit ausgeben, gibt es aber auch hierfür Möglichkeiten diese zu berechnen (siehe Platt, 2000).
Abbildung 9 zeigt die ROC-Kurven für die Datensituation S1. Diese zeigt grafisch noch einmal den deutlichen Unterschied zwischen *K-NN*, welcher nur etwas besser als eine reine Zufallsauswahl ist und den anderen vier Algorithmen, welche nahezu perfekt klassifizieren. Im Bezug auf **H2** sieht die Beweislage also eher dürftig aus, da *SVM-L* durschschnittlich in den linearen Szenarien eher im Mittelfeld rangiert.

In den Datensituationen mit polynomialer Form der Entscheidungsgrenze ist die Leistungsfähigkeit aller Algorithmen grundsätzlich schlechter als in den zuvor beschriebenen Szenarien. Im Fall von $p \ll n$ sind die Ergebnisse ausschließlich für *SVM-R* sehr gut mit Werten über 0.9. Die restlichen Algorithmen performen

	linear				**polynomial**				**radial**			
		ACC	AUC	F1	Rang		ACC	AUC	F1	Rang		ACC	AUC	F1	Rang
$p \ll n$	SVM-L	0.964	0.991	0.964	2	SVM-L	0.759	0.802	0.752	4	SVM-L	0.497	0.495	0.504	5
	SVM-P	0.964	0.991	0.964	2	SVM-P	0.764	0.802	0.756	2	SVM-P	0.926	0.981	0.926	1
	SVM-R	0.961	0.990	0.961	4	SVM-R	0.904	0.972	0.904	1	SVM-R	0.792	0.888	0.801	2
	LogR	0.965	0.991	0.965	1	LogR	0.762	0.802	0.751	3	LogR	0.500	0.500	0.667	4
	K-NN	0.571	0.600	0.567	5	K-NN	0.756	0.800	0.756	5	K-NN	0.687	0.749	0.657	3
		ACC	AUC	F1	Rang		ACC	AUC	F1	Rang		ACC	AUC	F1	Rang
$p \approx n$	SVM-L	0.80	0.851	0.783	3	SVM-L	0.60	0.590	0.524	3	SVM-L	0.58	0.531	0.462	5
	SVM-P	0.80	0.851	0.783	3	SVM-P	0.60	0.590	0.524	3	SVM-P	0.84	0.830	0.826	1
	SVM-R	0.86	0.885	0.851	1	SVM-R	0.60	0.586	0.545	2	SVM-R	0.74	0.889	0.794	2
	LogR	0.80	0.858	0.783	2	LogR	0.56	0.437	0.500	5	LogR	0.58	0.699	0.364	4
	K-NN	0.70	0.698	0.681	5	K-NN	0.62	0.729	0.642	1	K-NN	0.80	0.800	0.762	3
		ACC	AUC	F1	Rang		ACC	AUC	F1	Rang		ACC	AUC	F1	Rang
$p \gg n$	SVM-L	0.66	0.725	0.653	4	SVM-L	0.48	0.485	0.480	4	SVM-L	0.50	0.570	0.390	4
	SVM-P	0.70	0.750	0.681	3	SVM-P	0.50	0.517	NaN	5	SVM-P	0.76	0.726	0.750	1
	SVM-R	0.64	0.500	0.609	5	SVM-R	0.56	0.500	0.450	3	SVM-R	0.66	0.757	0.738	3
	LogR	0.72	0.754	0.741	2	LogR	0.64	0.642	0.654	2	LogR	0.54	0.538	0.343	5
	K-NN	0.80	0.866	0.808	1	K-NN	0.70	0.700	0.706	1	K-NN	0.76	0.760	0.700	2

Tabelle 2: Vergleich der Modelle

mittelmäßig mit Werten zwischen 0.7 und 0.8, wobei sich hier kein Algorithmus von den anderen absetzen kann. Bei $p \approx n$ ist *K-NN* den anderen Klassifikationsmethoden leicht überlegen, bei einem AUC-Wert von 0.729. Die anderen Algorithmen befinden sich durchweg über alle Werte hinweg nahe 0.5. Ähnlich ist das der Fall für $p \gg n$ bei dem auch wieder *K-NN* eine leicht bessere Leistung zeigt mit einer Genauigkeit von 0.7, gefolgt von *LogR* mit 0.64. Dennoch ist für die letzten beiden Szenarien deutlich, dass kein Algorithmus gute Leistung zeigt.
Abbildung 10 zeigt die ROC-Kurven für Szenario 2. Aus dieser Abbildung ist ersichtlich, dass sich *SVM-R* deutlich von den anderen Algorithmen abheben kann, welche alle eine mittelmäßige Leistung zeigen. Wir können also hier keine bestätigung für **H2** finden, nach der wir eigentlich erwarten würden, dass *SVM-P* hier besser Klassifiziert.

In den Datensituationen mit radialer Form der Entscheidungsgrenze ist für den Fall $p \ll n$ eine eindeutige Unterscheidung zu treffen. Während *SVM-P*, *SVM-R* und *K-NN* gut klassifizieren, haben dabei *SVM-L* und *LogR* große Probleme und zeigen lediglich Werte nahe 0.5. Insbesondere *SVM-P* zeigt mit einem AUC-Wert von 0.981 eine sehr gute Performance. Weniger drastisch aber dennoch mit dem gleichen Resultat ist dies für $p \approx n$ der Fall. Während *SVM-P* im Vergleich zum vorherigen Szenario etwas schlechter performt, bleibt *SVM-R* nahezu identisch und *K-NN* kann sich sogar leicht steigern auf einen AUC-Wert von 0.8. *SVM-L* und *LogR* können sich leicht verbessern (beispielsweise mit einer Genauigkeit nahe 0.6), belegen dennoch weiterhin Rang vier und fünf. Für $p \gg n$ sind ebenfalls die beiden Gruppen zu differenzieren. Für *SVM-L* und *LogR* sind wie in Szenario 3 wieder Werte nahe 0.5 für die Genauigkeit erkennbar. *SVM-P* und *K-NN* performen in diesem Szenario am Besten. *SVM-R* fällt insbesondere mit einer Genauigkeit von 0.66 im Vergleich zu 0.76 etwas hinter den anderen beiden Algorithmen zurück.

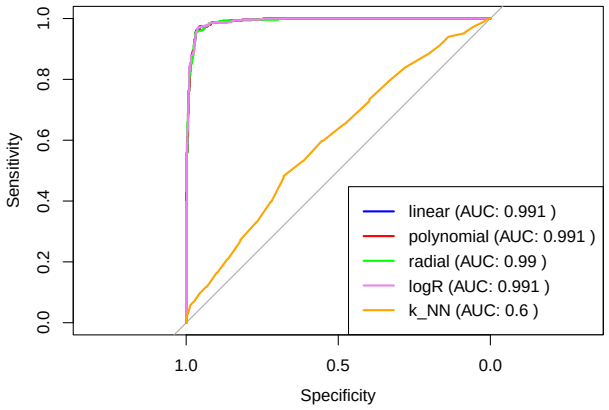

Abbildung 9: ROC-Kurven Szenario 1

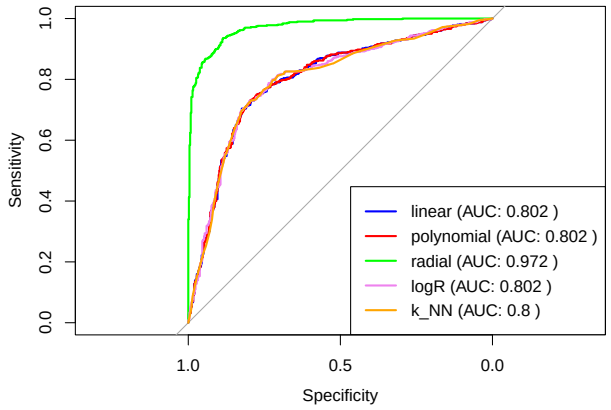

Abbildung 10: ROC-Kurven Szenario 2

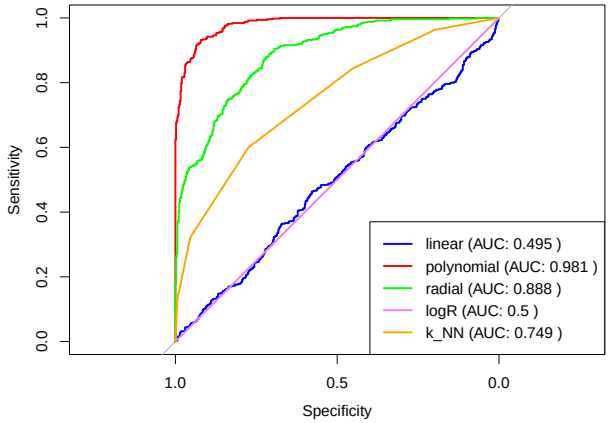

Abbildung 11: ROC-Kurven Szenario 3

Abbildung 11 zeigt die ROC-Kurven für Szenario 3. Hieraus wird deutlich, dass die Kurve von *SVM-P* nahe an der oberen, linken Ecke verläuft, was für eine nahezu perfekte Klassifikation spricht. Der etwas flachere Verlauf von *SVM-R* und *K-NN* macht deutlich, dass diese beiden Algorithmen weniger gute Leistung zeigen. Dennoch können die beiden oben erwähnten Gruppen gut identifiziert werden, da sich *SVM-L* und *LogR* nahe der diagonalen Linie orientieren. Diese beschreibt eine ROC-Kurve einer reinen Zufallsauswahl, woraus gefolgert werden kann, dass sie kaum Fähigkeit aufweisen zwischen den Klassen zu unterscheiden. Diese Szenarien bestätigen die Vermutung aus **H2** zwar nicht direkt, da hier im Ranking über alle Szenarien mir radialer Entscheidungsgrenze hinweg interessanterweise *SVM-P* am besten performt aber zumindest liegt hier *SVM-R* auf dem zweiten Platz. Trotzdem müssen wir mit diesen Ergebnissen **H2** ablehnen.

Zuletzt wird ein Vergleich der einzelnen Dimensionierungen über die drei Formen der Entscheidungsgrenze gezogen. So ist für $p \ll n$ ersichtlich, dass insbesondere *SVM-P* und *SVM-R* über alle drei Szenarien gute Leistung (mit Werten mindestens nahe 0.8) erzielen. Auch im durchschnittlichen Ranking über alle drei Szenarien hinweg zeigen die *SVM-Methoden* die besten Ergebnisse. Überaschen ist nur die unerwartet schlechte Performance von *K-NN*. Somit bestätigt sich **H3** hier nur teilweise. In den hochdimensionalen Szenarien $p \gg n$ ist einzig *K-NN* überzeugend, welcher auch im polynomialen Szenario Werte über 0.7 aufweist. Hier belegen die *SVM-Methoden* tatsächlich sogar die letzten Plätze im durchschnittlichen Ranking. Dies deutet klar darauf hin, dass **H4** mit unseren Daten verworfen werden muss.

7 Fazit

In unserer Arbeit haben wir zum einen die Funktionsweise der *Support Vector Machines* als binäre Klassifikationsmethode beleuchtet, als auch die Leistungsfähigkeit in Datenszenarien mit verschiedenen Eigenschaften verglichen. Diese Datensszenarien, wurden synthetisch von uns hergestellt und sind in ihrem datengenerierenden Prozess speziell auf *SVM* zugeschnitten. Anschließend haben wir uns mit einschlägiger Literatur beschäftigt, bei der die Autoren bereits ähnliche Versuche durchgeführt haben und anhand dessen Hypothesen abgeleitet. Für die Durchführung unserer Versuche haben wir die Performance von *SVM* mit verschiedenen Kernels sowie weiterer Klassifikationsmethoden über alle Szenarien mit verschiedenen Maßzahlen verglichen. Insgesamt mussten wir allerdings festellen, dass unsere Ergebnisse die Hypothesen nur in Teilen bestätigen. *SVM*-Methoden, haben im Durschnitt oft bessere Leistungen gezeigt, als die anderen Methoden. Wir schließen aufgrund der Rankings, dass, egal welche Dimensionierung vorliegt und wenn keine Information über die Form der Entscheidungsgrenze vorliegt, *SVM* mit radialen und polynomialen Kernel immer eine gute Entscheidung darstellen. Zumindest sollten diese beiden dem *SVM* mit linearem Kernel vorgezogen werden. Wobei hier auch anzumerken ist, dass mit der höheren Flexibilität der *SVM-P* und *SVM-R* auch die Gefahr besteht, dass es zu Overfitting kommt. Dies könnte vor allem bei empirischen Daten zu höheren Klassifikationsfehlern führen.
Allerdings haben wider Erwarten die *SVM* mit dem Kernel der auf die Datenerzeugung eigentlich zugeschnitten ist nicht besser performt. Wir vermuten, dass es an einem zu niedrigen n-Wert in der Dimensionierung für die Szenarien 4 bis 9 liegen könnte und dass sich in einer etwaigen Simulationsstudie mit dem gleichen DGP und Dimensionierung die Hypothese vielleicht doch noch bestätigen könnte, da sich so zufällige Abweichungen aufheben. Was das Verhalten in den verschiedenen Dimensionalitäten angeht können wir zumindest sagen, dass in Fällen mit mehr Beobachtungen als Variablen *SVM* eine gute Wahl darstellen. Schwieriger wird es bei den hochdimensionalen Szenarien, da hier *K-NN* eindeutig besser performt hat. Uns fällt es schwer dies zu erklären, da wir eigentlich davon ausgingen, dass *K-NN* bei vielen Variablen eher schlechter performt.

Wir sehen daher Optimierungsmöglichkeiten für weitere Arbeiten dieser Art. So wäre es angebracht, wie bereits erwähnt die Datengererierung für die einzelnen Szenarien wiederholt durchzuführen und die Ergebnisse zu mitteln. Die Dimesnionierung könnte auch so angepasst werden, dass die n Werte etwas höher sind auch für Szenarien mit $p \gg n$.
Des Weiteren könnte der Einfluss der Szenarien auf die Berechnungszeit für die *SVM*-Algorithmen noch einbezogen werden. Ein Benchmark-Test könnte auch hier zu interessanten Ergebnissen führen. Zusätzlich haben wir hier lediglich eine handvoll Klassifikationsalgorithmen im Vergleich untersucht. Eine Erweiterung auf *Classification Trees*, *Discriminant Analysis* oder verschiedene *Ensemble-Methoden* ist denkbar. An der

Datengenerierung ließen sich ebenfalls weitere Aspekte anpassen. So könnte die Anteile der Ausprägungen in der binären Zielvariable noch variiert, mehr als zwei Ausprägungen generiert oder auch komplexere Entscheidungsgrenzen modelliert werden. Diese Erweiterungen hätten allerdings den Rahmen dieser Arbeit überschritten. Trotzdem ergänzt unsere Arbeit die bisherigen Befunde zur Leistungs- und Anpassungsfähigkeit von *Support-Vector Machines* in verschiedenen Datensituation, sowie deren Bedeutung im Kontext von Klassifikationsaufgaben.

Literatur

Bennett, K., & Campbell, C. (2000). Support Vector Machines: Hype or Hallelujah? *ACM SIGKDD Explorations Newsletter*, *2*, 1–13. https://doi.org/10.1145/380995.380999

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, 144–152. https://doi.org/10.1145/130385.130401

Burges, C. J. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, *2*(2), 121–167. https://doi.org/10.1023/A:1009715923555

Cervantes, J., Garcia-Lamont, F., Rodríguez-Mazahua, L., & Lopez, A. (2020). A Comprehensive Survey on Support Vector Machine Classification: Applications, Challenges and Trends. *Neurocomputing*, *408*, 189–215. https://doi.org/10.1016/j.neucom.2019.10.118

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297. https://doi.org/10.1007/BF00994018

Entezari-Maleki, R., Rezaei, A., & Minaei-Bidgoli, B. (2009). Comparison of Classification Methods Based on the Type of Attributes and Sample Size. *Journal of Convergence Information Technology*, *4*(3), 94–102. https://doi.org/10.4156/jcit.vol4.issue3.14

Fávero, L. P., Belfiore, P., Santos, H. P., dos Santos, M., de Araújo Costa, I. P., & Junior, W. T. (2022). Classification Performance Evaluation from Multilevel Logistic and Support Vector Machine Algorithms through Simulated Data in Python. *Procedia Computer Science*, *214*, 511–519. https://doi.org/10.1016/j.procs.2022.11.206

Fawcett, T. (2006). An Introduction to ROC Analysis. *Pattern Recognition Letters*, *27*(8), 861–874. https://doi.org/10.1016/j.patrec.2005.10.010

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An Introduction to Statistical Learning: With Applications in R*. Springer US. https://doi.org/10.1007/978-1-0716-1418-1

Kecman, V. (2005, 22. April). Support Vector Machines – An Introduction. In L. Wang (Hrsg.), *Support Vector Machines: Theory and Applications* (S. 1–47, Bd. 177). Springer Berlin Heidelberg. https://doi.org/10.1007/10984697_1

Kuhn, M., & Johnson, K. (2013). *Applied Predictive Modeling*. Springer New York. https://doi.org/10.1007/978-1-4614-6849-3

Mavroforakis, M. E., & Theodoridis, S. (2006). A Geometric Approach to Support Vector Machine (SVM) Classification. *IEEE transactions on neural networks*, *17*(3), 671–682. Zugriff 28. August 2024 unter https://ieeexplore.ieee.org/abstract/document/1629090/

Moguerza, J. M., & Muñoz, A. (2006). Support Vector Machines with Applications. Zugriff 27. August 2024 unter https://projecteuclid.org/journals/statistical-science/volume-21/issue-3/Support-Vector-Machines-with-Applications/10.1214/088342306000000493.short

Platt, J. (2000). Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. *Adv. Large Margin Classif.*, *10*.

Scholz, M., & Wimmer, T. (2021). A Comparison of Classification Methods across Different Data Complexity Scenarios and Datasets. *Expert Systems with Applications*, *168*, 114217. https://doi.org/10.1016/j.eswa.2020.114217

Snoek, J., Larochelle, H., & Adams, R. P. (2012, 29. August). *Practical Bayesian Optimization of Machine Learning Algorithms*. arXiv: 1206.2944 [cs, stat]. Zugriff 19. August 2024 unter http://arxiv.org/abs/1206.2944

Soofi, A. A., & Awan, A. (2017). Classification Techniques in Machine Learning: Applications and Issues. *Journal of Basic & Applied Sciences*, *13*, 459–465. Zugriff 27. August 2024 unter http://set-publisher.com/index.php/jbas/article/view/1715

Vapnik, V. (2006). *Estimation of Dependences Based on Empirical Data*. Springer. https://doi.org/10.1007/0-387-34239-7

Yang, L., & Shami, A. (2020). On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice. *Neurocomputing*, *415*, 295–316. https://doi.org/10.1016/j.neucom.2020.07.061