technische universität
dortmund

Bachelorarbeit

Effiziente kernelbasierte Klassifikation von Teleskop-Daten durch die Anwendung der Nyström Approximation

Martin Senz
Oktober 2018

Gutachter:
Prof. Dr. Katharina Morik
Lukas Pfahler

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Künstliche Intelligenz (LS-8)
https://www-ai.cs.uni-dortmund.de/

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Effizientes kernelbasiertes Lernen

Viele Anwendungen des maschinellen Lernens erfordern das Verarbeiten von großen, nicht linear-trennbaren, Datensätzen. Ein Beispiel für einen solchen Anwendungsbereich ist die häufig eingesetzte Mustererkennung in Bilddaten. In der Regel sind Künstliche Neuronale Netze für solche Klassifikationsaufgaben prädestiniert und stellen die Wahl der Mittel dar. Sollen zur Klassifizierung auf lineare Methoden, wie die Support Vector Machines (SVM), zurückgegriffen werden, so ist der direkte Einsatz auf nicht linear-separierbaren Daten häufig nicht erfolgversprechend. Damit SVM sinnvoll auf nicht linear-trennbaren Daten angewendet werden kann, werden häufig Kernel Methoden verwendet, die den Anwendungsbereich von SVM auf nicht linear-trennbaren Daten erweitern.
Allerdings entsteht durch den Einsatz von Kernel Methoden ein erhöhter Ressourcenverbrauch, der stark von der Eingabegröße abhängt. Falls auf großen Datensätzen gelernt werden soll, wird der Gebrauch von Kernel SVM schnell ineffizient und nicht praktikabel. Das Lernen von sehr vielen Daten ermöglicht häufig akkuratere Vorhersagen von Datenbeispielen. Folglich besteht die Anforderung, effizient und kernelbasiert, auf vielen Daten trainieren zu können. Eine Anwendung, die effizientes Lernen mit Kernel Methoden erfordert, ist das maschinelle Lernen von Teleskop-Daten.
In dieser Arbeit werden Methoden vorgestellt, die kernelbasiertes Lernen für große Datensätze skalierbar halten. Dabei wird der zentrale Ansatz zur Reduzierung des Ressourcenverbrauchs, auf die Approximation der Kernel Matrix beruhen. Die Methoden werden mit dem Ziel vorgestellt, Kernel SVM erfolgreich auf FACT Teleskop-Daten anzuwenden.

1.2 Aufbau und Ziele der Arbeit

Die Arbeit besteht aus zwei inhaltlichen Teilen. Im ersten Teil werden die zum Einsatz kommenden Methoden vorgestellt. Zunächst wird die untersuchte FACT Lernaufgabe be-

trachtet und formal definiert. Anschließend werden die notwendigen theoretischen Grundlagen zum Lernen mit Kernel Methoden gelegt, sowie die resultierenden Effizienzprobleme diskutiert. Als zentrale Lösungsmethode wird ein Verfahren vorgestellt, das unter Anwendung der Nyström Approximation, linearisierte Modelle generiert. Die Methoden werden auf dem MNIST und FACT Datensatz getestet, wobei die erreichte Performanz und der notwendige Aufwand analysiert werden.
Weitergehend wird im zweiten Teil der Bachelorarbeit die Genauigkeit von approximierten Matrizen untersucht. Insbesondere werden Funktionsabtastungen innerhalb von Kernel Methoden betrachtet. Die Abtastungen von Kernel Methoden üben einen interessanten Einfluss auf die Eigenwerte der resultierenden Matrizen aus, der es ermöglicht, rang-reduzierte Matrizen zu induzieren. Im abschließenden Teil der Arbeit wird experimentell evaluiert, ob die generierten rang-reduzierten Matrizen das Potential besitzen, eine genauere und effizientere Anwendung der Nyström Approximation zu ermöglichen.

Kapitel 2

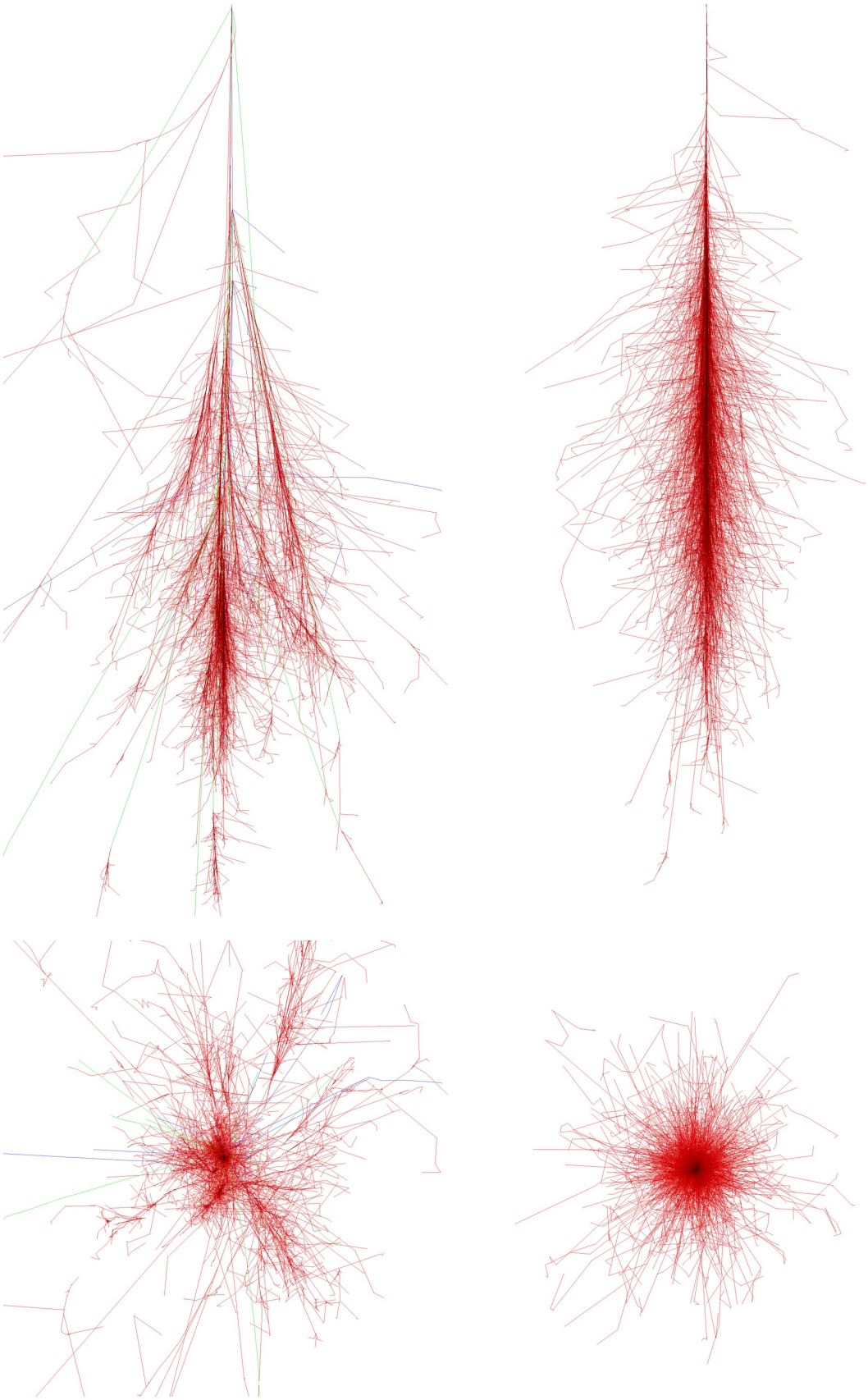
Anwendung

Dieses Kapitel dient zur Vorstellung der betrachteten FACT-Experimente sowie des MNIST Datensatzes. Beide Anwendungen bestehen aus der Anforderung Daten korrekt in Gruppen zu klassifizieren. Für diesen Zweck werden maschinelle Methoden der Klassifikation verwendet.

2.1 FACT

Die FACT-Experimente sind dem physikalischen Forschungszweig der Gammastrahlen Astronomie zuzuordnen, die sich primär mit der Detektion und Erforschung von kosmischer Gammastrahlung beschäftigt. Im Universum existieren zahlreiche Quellen, die Gammastrahlung erzeugen, wie beispielsweise aktive galaktische Kerne oder die Überreste von Supernoven. Wird Gammastrahlung detektiert und analysiert, so ist es möglich die kosmische Quelle der Strahlung zu rekonstruieren, wobei sich diese in fernen Galaxien befinden kann. Daher besitzt die Analyse von kosmischer Gammastrahlung das Potenzial, Rückschlüsse über extragalaktische Objekte zu ziehen. Demnach dient die Gammastrahlen Astronomie der weiteren Erforschung des Universums.
Allerdings stellt die Detektion von kosmischer Gammastrahlung eine Herausforderung in der Gammastrahlen Astronomie dar [6]. Trifft Gammastrahlung auf die Erde, so werden die hoch energetischen Photonen der Gammastrahlung von der Atmosphäre absorbiert. Eine direkte, erdgebundene, Beobachtung von Gammastrahlung ist somit nicht möglich. Allerdings entsteht eine Wechselwirkung zwischen den auftreffenden Teilchen und der Erdatmosphäre: Ein Luftschauer aus Sekundärteilchen bildet sich, der in kaskadierender Form zur Erde fällt. Die zur Erde fallenden Teilchen werden stark beschleunigt und erfüllen damit den sogenannten Cherenkov-Effekt. Dies hat zur Folge, dass die Teilchen für kurze Zeit Lichtblitze, das sogenannte Cherenkov-Licht, erzeugen. Von der Erde ausgehend kann das Cherenkov-Licht mit speziellen Teleskopen, den bildgebenden Cherenkov-Teleskopen (**I**maging **A**tmospheric **C**herenkov **T**elescope, **IACT**), detektiert werden. Folglich ist es

über das Observieren des Nachthimmels mit **IACT** Teleskopen möglich, kosmische Gammastrahlung indirekt zu detektieren.
Das FACT-Teleskop (**F**irst G-**A**PD **C**herenkov **T**elescope) gehört zu der Klasse der bildgebenden Cherenkov-Teleskopen. Das durch Luftschauer entstehende Cherenkov-Licht wird von einer Kamera mit 1440 Pixeln für eine kurze Zeitsequenz aufgenommen. Die entstehenden Aufnahmen der FACT-Kamera spiegeln die Häufigkeit der gemessenen Photonenhäufigkeit in jedem Pixel wieder. [2]
Eine große Schwierigkeit beim Detektieren von kosmischer Gammastrahlung ergibt sich aus dem Umstand, dass der Cherenkov-Effekt nicht exklusiv für Luftschauer gilt, die durch Gammaquanten ausgelöst wurden. Durch Protonen ausgelöste Luftschauer erfüllen ebenfalls den Cherenkov-Effekt und erzeugen Lichtblitze, die von Cherenkov-Teleskopen beobachtet werden. Dies hat zur Folge, dass bei der Auswertung der FACT-Aufnahmen nicht trivialerweise zugeordnet werden kann, ob der observierte Luftschauer durch Gamma oder Protonstrahlung ausgelöst wurde. Da in der Regel nur physikalisches Interesse an Gamma-induzierte Luftschauer besteht, folgt daraus die Anforderung Gamma-Luftschauer korrekt von Protonen-induzierten Luftschauern zu separieren.
Damit Luftschauer korrekt vorhergesagt werden können, müssen charakteristische Eigenschaften zwischen Gamma- und Proton-induzierten Luftschauern vorhanden sein. Dies ist bei der Gamma/Proton Separation durch ein unterschiedliches Flugverhalten der Gamma- und Proton-Luftschauer gegeben. Gamma-Luftschauer weisen häufig einen dichteren Kern aus Teilchen auf. Die Teilchen von Proton-Luftschauern fallen weniger Dicht und großflächiger zur Erde. Das unterschiedliche Flugverhalten der Teilchen manifestiert sich auch bei den FACT-Aufnahmen: es entstehen zum Flugverhalten korrespondierende Muster. Diese Muster können von maschinellen Methoden gelernt werden und für Vorhersagen über unbekannte Luftschauer verwendet werden. Dabei werden Muster von simulierten Luftschauer gelernt, welche auf Grundlage von Monte Carlo Simulationen erstellt wurden.
Allerdings stellt die Separation von Gamma- und Proton-Luftschauern eine anspruchsvolle Aufgabe dar. Die Muster lassen sich in Fällen nicht eindeutig mit einer Klasse assoziieren, so dass die korrekte Vorhersage von Luftschauern erschwert wird. Des Weiteren wird durch die Photonensensibilität des FACT-Teleskops und die Wirkung des Cherenkov-Effektes, FACT-Aufnahmen durch ein Grundrauschen aus detektierten Photonen überlagert. Es sind aufwändige Datenaufbereitungsschritte notwendig, um relevante Muster der FACT-Aufnahmen zu extrahieren.



(c) Proton Luftschauer (d) Gamma Luftschauer

Abbildung 2.1: Simulierte Luftschauer

2.2 MNIST

Der MNIST Datensatz stammt aus den Bereich der Handschrifterkennung und dient zur Klassifizierung von handschriftlichen Ziffern zwischen 0 und 9. Die Handschrifterkennung stellt ein klassischen Anwendungsbereich im maschinellen Lernen dar. Durch das frequentierte Aufkommen von Touch- und Stifteingaben gehört die Handschrifterkennung häufig im Repertoire von Anwendungssoftware. Jedes Datenbeispiel aus dem MNIST Datensatz besteht aus einem Pixelbild mit 28×28 Pixel, die jeweils eine handschriftliche Ziffer repräsentieren. Der Datensatz wurde bereits ausgiebig mit unterschiedlichen Klassifikatoren und Methoden erforscht. Daher soll er primär dazu dienen, die Leistungsfähigkeit der vorgestellten Methoden einzuschätzen.

Kapitel 3

Statistische Lerntheorie

Im vorherigen Kapitel wurde informell die durchzuführende Klassifizierung der FACT und MNIST Anwendung beschrieben. In diesem Kapitel werden die notwendigen Konzepte vorgestellt, um die Lernaufgabe formal zu definieren. Wie man sehen wird, lässt sich die Lernaufgabe nach den Prinzipien der statistischen Lerntheorie definieren.

3.1 Überwachtes Klassifizieren

Bei den hier betrachteten Anwendungsfällen handelt es sich um überwachtes Lernen. Es wird in zwei Phasen unterschieden: einer Lernphase und einer Testphase.
In der Lernphase wird auf Grundlage von Trainingsdaten eine Hypothese h_ℓ durch einen Lerner ℓ induktiv abgeleitet. Sei X die Datenbeispiele und Y die Menge der Klassen, dann ist h_ℓ eine Funktion definiert durch $h_\ell : X \longmapsto Y$. Die Hypothese dient als Klassifizierungsregel (Modell) und soll die Klassenzugehörigkeiten von Datenbeispielen korrekt vorhersagen. Die Hypothese wird durch das Lernen auf einer Trainingsmenge gebildet, die aus Daten mit bekannter Klassifizierung besteht. Für n Trainingsbeispiele wird die Trainingsmenge durch

$$S = \{(x_i, y_i) | x_i \in X, y_i \in Y, i \in \{1, ...n\}\} \tag{3.1}$$

definiert. Jedes Trainingsbeispiel besitzt eine eindeutige Klasse.
In der Testphase wird geprüft, wie leistungsfähig h_ℓ bezüglich korrekter Vorhersagen von Datenbeispielen ist. Hierbei wird unter Verwendung von h_ℓ Vorhersagen von Datenbeispielen aus einer Testmenge getätigt. Da die tatsächlichen Klassenzugehörigkeiten der Testdaten gegeben sind, lassen sich mittels geeigneter Metriken die Leistungsfähigkeit des Klassifikators einschätzen.

3.2 Minimieren des Risikos

Die Grundlage für die Modellerstellung h_ℓ stellt die Minimierung der *Risk* Funktion dar. Unter der Annahme, dass die Trainingsdaten $(x_1, y_1)...(x_n, y_n)$ unabhängig und identisch verteilt aus einer Verteilung $\mathbf{P}(x, y)$ über X und Y gezogen wurden, ist die Risk Funktion definiert durch

$$R(h_\ell) = \mathbf{E}[L(h_\ell(x), y)] = \int L(h_\ell(x), y) dP(x, y) \tag{3.2}$$

wobei L eine Loss-Funktion darstellt. Eine Loss Funktion $L(\hat{y}, y)$ bewertet und quantifiziert die Korrektheit einer Vorhersage $\hat{y}$ mit dessen tatsächlicher Klassenzugehörigkeit y. Ein elementares Beispiel einer Loss Funktion ist das L_{0-1} Loss mit $L_{0-1}(\hat{y}, y) = \mathbf{I}(\hat{y} = y)$, wobei $\mathbf{I}$ die Indikatorfunktion darstellt. Demnach ist die Risk Funktion der Erwartungswert der Loss Funktion, welcher unter h_ℓ auf dem Trainingsdatensatz entsteht. Folglich ist das Ziel eine Hypothese h_ℓ zu finden, für das 3.2 minimal wird.
Allerdings lässt sich die Risk Funktion im Allgemeinen nicht berechnen, da die Verteilung $\mathbf{P}(x, y)$ in der Regel unbekannt ist. Eine Lösung ist die Approximation der Risk Funktion durch einen Schätzer - der empirischen *Risk* Funktion.
Die empirische Risk Funktion ist definiert durch

$$R_{emp}(h_\ell) = \frac{1}{n} \sum_{i=1}^{n} L(h_\ell(x_i), y_i). \tag{3.3}$$

Ein direktes minimieren der empirischen Risk Funktion ist problematisch, da die Gefahr der Überanpassung an den Trainingsdaten besteht. Bei einem überangepassten Modell, ist typischerweise der Trainingsfehler sehr niedrig, dass heißt 3.4 besitzt einen niedrigen Wert. Zugleich klassifiziert ein überangepasstes Modell viele Testdaten fehlerhaft, was sich durch einen hohen Testfehler (Generalisierungsfehler) ausdrückt. Liegt ein extremer Fall von Überanpassung vor, lernt ein Modell nur die Trainingsbeispiele mit zugehöriger Klasse auswendig. Dies hat zur Folge, dass unbekannte Beispiele durch den Lerner nicht zufriedenstellend klassifiziert werden können. Neben der fehlenden Generalisierung, führt das Auswendiglernen von Daten zu Modellen mit hoher Komplexität. Daher ist die Modellkomplexität ein Indiz, ob eine Überanpassung vorliegt und welche Qualität das Vorhersagemodell besitzt. Die Emprical Risk Minimzation (ERM) kann eine hohe Komplexität vermeiden, in dem h_ℓ aus einer begrenzten Menge von Funktionen stammt. Mit einem solchen, restriktiven, Hypothesenraum H wird das Minimierungsproblem

$$h^* = arg \min_{h_\ell \in H} R_{emp}(h_\ell) \tag{3.4}$$

gelöst. Neben der Begrenzung der zulässigen Hypothesen, lässt sich Überanpassung vermeiden, wenn komplexe Modelle bestraft werden. Dieser Ansatz wird von der Structural

Risk Minimization (SRM), beziehungsweise durch den Einsatz von Regularisierung verfolgt [27]. Das Minimieren der regularisierten Risk Funktion ist durch

$$h^* = arg \min_{h_\ell \in H} R_{emp}(h_\ell) + \lambda R(h_\ell) \tag{3.5}$$

definiert, wobei R die Regularisierung darstellt. Der entstehende Zielkonflikt zwischen Modellkomplexität und Minimierung des Trainingsfehlers, wird bei der regularisierten empirischen Risk Funktion durch λR gesteuert.

3.3 Lernaufgabe

Die Klassifikation der Gamma- und Proton-Luftschauer wird durch das Minimieren der regularisierten empirischen Risk Funktion gelöst. Das Ziel ist ein Modell h_ℓ zu erzeugen, sodass FACT Aufnahmen bezüglich ihrer Klasse durch h_ℓ korrekt vorhersagt werden. Jede FACT Aufnahme wird durch einen Datenvektor $x \in \mathbb{R}^{1440}$ repräsentiert, der die gemessene Photonenhäufigkeit in jedem Pixel der Kamera widerspiegelt. Jedes Datenbeispiel x gehört entweder zur Proton- oder Gamma-Klasse. Entsprechend gilt $Y = \{-1 \text{ (Proton)}, 1 \text{ (Gamma)}\}$. Da simulierte Datenbeispiele zum Training und Testen verwendet werden, sind die tatsächlichen Klassenzugehörigkeiten der Daten bekannt. Daher lässt sich die Trainingsmenge S und die Testmenge definieren. Folglich lässt sich ein Vorhersagemodell durch 3.5 ableiten.
Analog zur FACT Anwendung wird die Lernaufgabe für den MNIST Datensatz definiert. Der MNIST Datensatz besteht aus Bilddaten mit 28×28 Pixel. Daher gilt für jedes Datenbeispiel $x \in \mathbb{R}^{784}$. Jedes MNIST Bild ist genau einer Ziffer zugeordnet. Somit gilt für $Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Kapitel 4

Support Vector Machines

Wie bereits dargelegt wurde, lassen sich die definierten Lernaufgaben der FACT und MNIST Anwendung nach dem Prinzipien der statistischen Lerntheorie definieren. Eine Realisierung der Minimierung der regularisierten empirischen Risk Funktion stellen die Support Vector Machines dar.

4.1 Klassifizieren von linear separierbaren Daten

Support Vector Machines konstruieren auf Grundlage von Trainingsdaten eine Hyperebene, welche die Trainingsdaten optimal trennt. Die in der Lernphase konstruierte Hyperebene wird verwendet, um Vorhersagen über Datenbeispiele zu geben. Zunächst wird der Fall von linear separierbaren Daten betrachtet, die mittels SVM klassifiziert werden sollen.
Seien (x_1, y_1) ... (x_n, y_n) lineare Trainingsdaten mit $x \in \mathbb{R}^m$ und $y \in \{-1, 1\}$. Dann existiert eine Hyperebene $\langle w, x\rangle + b = 0$, welche die Trainingsbeispiele fehlerfrei separiert. Die Vektoren, die am nächsten zur Hyperebene liegen, werden Support Vektoren genannt und besitzen zur Hyperebene genau den Abstand δ.
Eine optimale Hyperebene wird charakterisiert durch:

- Alle Trainingsdaten werden korrekt durch die Hyperebene getrennt.
- Der Abstand δ zwischen Support Vektoren und Hyperebene wird maximiert.

Die Konstruktion der Hyperebene hängt ausschließlich von den Support Vektoren ab. Wenn alle Trainingsdaten korrekt getrennt wurden, gilt für jedes Trainingsbeispiel x_i:

$$\langle w, x_i\rangle + b \geq 1, y_i = 1 \text{ (Positives Beispiel)} \tag{4.1}$$

$$\langle w, x_i\rangle + b \leq -1, y_i = -1 \text{ (Negatives Beispiel)} \tag{4.2}$$

Zusammengefasst lassen sich 4.1 und 4.2 zu

$$y_i[\langle w, x_i\rangle + b] \geq 1 \tag{4.3}$$

formalisieren. Folglich drückt 4.3 die Forderung aus, dass alle Trainingsdaten korrekt separiert wurden.
Der zu maximierende Abstand δ ist durch $\delta = \frac{2}{\|w\|}$ definiert [28]. Die resultierende Maximierungsaufgabe von $\delta = \frac{2}{\|w\|}$ kann äquivalent zu $\min_{\mathbf{w}} \|\mathbf{w}\|$ umgewandelt werden. Daraus folgt die Konstruktion der optimalen Hyperebene als Optimierungsproblem durch:

$$\begin{aligned} &\text{Minimiere } \|\mathbf{w}\| \\ &\quad \text{Unter der Nebenbedingung: } y_i[\langle w, x_i\rangle + b] \geq 1, \text{ für alle } i \in [1, ..., n] \end{aligned} \tag{4.4}$$

4.2 Soft-Margin SVM

Das in 4.4 definierte Problem, ist als Hard-Margin Optimierungsproblem für SVM bekannt [28]. Die Restriktion 4.3 erzwingt, dass alle Trainingsdaten korrekt klassifiziert werden. Diese Bedingung ist für viele Anwendungsfälle zu strikt, da viele Datensätze nicht vollständig linear trennbar sind. Stattdessen ist eine kontrollierbare Fehlertoleranz beim Training wünschenswert.
Zu diesem Zweck wird die Hinge Loss Funktion für ein Datenbeispiel x mit $L_{HINGE}(x) = max(1 - y(\langle w, x\rangle + b), 0)$ eingeführt. Das Hinge Loss ist eine konvexe Approximation der $L_{0,1}$ Funktion. Werden Datenbeispiele korrekt klassifiziert, dann beträgt der Wert der Hinge Loss Funktion null. Verletzt ein Datenbeispiel den Abstand δ oder wird es durch die Hyperebene falsch klassifiziert, dann spiegelt das Hinge Loss den Grad des Fehlers wieder. Die sogenannte Soft Margin Definition von SVM führt eine Schlupfvariable ξ mit $\xi_i = L_{HINGE}(x_i) = max(1 - y(\cdot\langle w, x_i\rangle + b), 0)$ ein [28]. Das Primal Soft-Margin Problem wird formuliert zu:

$$\begin{aligned} &\text{Minimiere } \frac{1}{n}\sum_{i=1}^{n} \xi i + \lambda\|\mathbf{w}\|^2 \\ &\quad \text{Unter der Nebenbedingung: } y_i[\langle w, x_i\rangle + b] \geq 1 - \xi_i, \text{ für alle } i \in [1, ..., n] \end{aligned} \tag{4.5}$$

Die Summe von ξ_i repräsentiert den Trainingsfehler und $\|\mathbf{w}\|^2$ dient als $L2$ Regularisierung. Offensichtlich beeinflusst λ die Komplexität der Hyperebene. Ein hoher λ Wert bestraft komplexe Hyperebenen (Hyperebenen mit großen Koeffizienten), was zur Folge hat, dass der Trainingsfehler steigt. Analog sinkt der Trainingsfehler und steigt die Modellkomplexität, wenn λ klein gewählt wurde.
Das in 4.5 definierte Optimierungsproblem lässt sich ohne Nebenbedingung formulieren zu:

$$\text{Minimiere } \frac{1}{n}\sum_{i=1}^{n} max(1 - y(\langle w, x_i\rangle + b), 0) + \lambda\|\mathbf{w}\|^2 \tag{4.6}$$

Offensichtlich minimiert 4.6 die regularisierte Risk Funktion 3.5 für $L = L_{HINGE}$. Die zu klassifizierenden Beispiele werden mit einer Entscheidungsfunktion klassifiziert, die durch die konstruierte Hyperebene gegeben ist. Ein Beispiel x wird durch

$$h(x) = sgn(\langle w, x \rangle + b) \tag{4.7}$$

vorhergesagt.

4.3 Multiclass-Klassifikation

Die hier definierten Support Vector Machines sind ausschließlich für die binäre Klassifikation ($|Y| = 2$) definiert. Für den MNIST Datensatz besteht die Anforderung der Klassifikation für $|Y| = 10$. Entsprechend ergibt sich die Notwendigkeit SVM um Multiklassen Klassizität zu erweitern.
Ein populärer Ansatz ist die Reduzierung eines Multiklassen Problems in binäre Klassifikationsaufgaben. Im Kontext dieser Arbeit wird die häufig gewählte One vs All Strategie (auch One vs Rest genannt) verwendet. Bei der One vs All Strategie existiert für jede Klasse genau ein Klassifikator. Der jeweilige Klassifikator interpretiert alle Trainingsbeispiele die zu seiner korrespondierenden Klasse gehören, als positive Beispiele. Die restlichen Beispiele werden als negative Beispiele eingestuft. Soll nun eine Vorhersage zu einem Objekt getätigt werden, bildet jeder Klassifikator den Konfidenzwert für das Objekt. Das Objekt wird für die Klasse vorhergesagt, dessen korrespondierender Klassifikator den höchsten Konfidenzwert ermittelt hat.

Kapitel 5

Kernelmethoden im maschinellen Lernen

Die FACT und MNIST Anwendung erfordern das Lernen auf nicht linear-separierbaren Daten. Soll die Klassifikation durch lineare Methoden gelöst werden, so werden meistens Kernelmethoden verwendet, die im Folgenden vorgestellt werden.

5.1 Einsatz von linearen Methoden

Die im vorherigen Kapitel eingeführten Support Vector Machines gehören zur Klasse der linearen Klassifizierungsmethoden. In der Regel basieren lineare Methoden auf gut erforschten und soliden mathematischen Prinzipien. Durch das mathematische Fundament wird häufig eine gute Analysierbarkeit der verwendeten Methoden erreicht, wodurch beispielsweise Fehlerschranken definiert werden können. So lassen sich für SVM Schranken für den Generalisierungsfehler definieren. [26] [24]
Eine weitere Stärke linearer Methoden ist das sehr effiziente Lernen von Daten. Ein Beispiel für die effiziente und erfolgreiche Anwendung von linearen Methoden, stellen die Support Vector Machines bei der Textklassifizierung dar [16]. Allerdings ist die erfolgreiche Anwendung sehr stark von der zugrunde liegenden Trennbarkeit der Daten abhängig. Bei der Textklassifizierung ist eine gute lineare-Trennbarkeit der Datenbeispiele gegeben, so dass mit kluger Parametersetzung sehr effizient große Mengen an Daten klassifiziert werden können. Anders sieht die Anwendung auf Bilddaten aus. Bilddaten können nicht linear getrennt werden. Eine direkte Anwendung von SVM auf dem FACT oder MNIST Datensatz sind daher nicht erfolgversprechend. Damit der Einsatz von SVM auf nicht linear-separierbaren Daten erweitert wird, werden Kernelmethoden verwendet.

5.2 Kernelmethoden

Die Grundlegende Idee von Kernelmethoden stellt das Abbilden von nicht linear-separierbaren Daten in einen höherdimensionalen Raum H dar. Falls H eine hinreichend große Dimensionalität aufweist, so sind die in H transformierten Datenbeispiele linear-trennbar und können mit linearen Methoden verarbeitet werden. Sei $\phi : X \longmapsto H$ eine Transformationsfunktion, die Datenbeispiele $x \in X$ mit $\phi(x)$ in H einbettet. In Abhängigkeit zur Dimensionalität des abgebildeten Raums H, kann der Aufwand ϕ zu berechnen sehr hoch sein. Da H potentiell eine unendlich große Dimensionalität aufweisen kann, ist die Berechnung der Transformationsfunktion in solchen Fällen nicht möglich.
Kernelmethoden lösen dieses Problem, durch die Vermeidung der expliziten Berechnung von ϕ. Dies wird erreicht, indem jede Kernelmethode zu einem Skalarproduktraum korrespondiert. Solange Algorithmen nur das Skalarprodukt in H berechnen, wird die, unter Umständen sehr teure, Transformationsfunktion nicht explizit berechnet.

Definition. Eine symmetrische Funktion $k : X \times X \longmapsto \mathbb{R}$ mit $(x, x') \mapsto k(x, x')$ heißt Kernel, falls für alle $x, x' \in X$

$$k(x, x') = \langle \phi(x), \phi(x') \rangle \tag{5.1}$$

erfüllt wird.

Für jedes $x, x' \in X$ müssen Kernelmethoden in einem wohldefinierten Skalarproduktraum abbilden. Da nur im Skalarproduktraum operiert wird, muss ϕ nicht explizit berechnet oder bekannt sein.
Kernelmethoden können als Ähnlichkeitsmaß zwischen den Datenbeispielen aufgefasst werden. Seien $x_1, \ldots, x_n \in X$ Datenvektoren, auf die eine Kernel-Funktion angewendet werden soll. Dann wird für jede paarweise Kombination aus $x_1, \ldots, x_n$ die Ähnlichkeit der Datenbeispiele mittels einer Kernelmethode bewertet. Es entsteht eine Kernel-Matrix.

Definition. Sei k eine Kernel-Funktion, $x_1, \ldots, x_n \in X$ und $i, j \in \{1, ..., n\}$.
Dann wird eine Matrix

$$K := (k(x_i, x_j))_{i,j} \tag{5.2}$$

Kernel-Matrix genannt.

Da $(k(x_i, x_j))_{i,j} = \langle \phi(x_i), \phi(x'_j) \rangle_{i,j}$ gilt, stellt eine Kernel-Matrix eine Grammsche Matrix dar. Folglich muss jede Kernel-Matrix positiv semidefinit (psd) sein.

5.3 Reproduziernde Kernel Hilberträume

Eine zusätzliche Perspektive auf Kernelmethoden liefert die Betrachtung von speziellen Skalarprodukträumen - den Hilberträumen mit reproduzierendem Kern (Reproducing Kernel Hilbert Spaces, RHKS). Nach dem Moore–Aronszajn Theorem induziert jede Kernelmethode einen RKHS [3]. Ausgehend von einer Kernelmethode lässt sich das RKHS konstruieren.
Sei k eine Funktion die 5.1 erfüllt, X die Menge der Datenbeispiele und $\phi : X \longmapsto \mathbb{R}^X$. Ein Objekt $x \in X$ wird mittels $\phi(x) = k(\cdot, x)$ eingebettet. Die lineare Hülle der eingebetteten Objekte $\phi(x)$ bilden einen Vektorraum $\mathcal{H}$ mit $\mathcal{H} = span(\{\phi(x) : x \in X\}) = \{f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\cdot, x_i) : m \in \mathbb{N}, x_i \in X, \alpha_i \in \mathbb{R}\}$. Sei $f, g \in \mathcal{H}$ mit $f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\cdot, x_i)$ und $g(\cdot) = \sum_{j=1}^{n'} \beta_j k(\cdot, x_j)$. Dann wird ein Skalarprodukt $\langle f, g \rangle$ definiert durch

$$\langle f, g \rangle := \sum_{i=1}^{n} \sum_{j=1}^{n'} \alpha_i \beta_j k(x_i, x_j). \tag{5.3}$$

Unter der Annahme, dass k eine Kernelmethode ist, lässt sich zeigen, dass 5.3 alle Eigenschaften eines Skalarproduktes aufweist und entsprechend $\mathcal{H}$ ein wohldefinierter Skalarproduktraum (Hilbertraum) ist (Siehe [14] für Nachweis). Insbesondere gilt die reproduzierende Eigenschaft mit $\langle k(\cdot, x), f \rangle = f(x)$ für alle $x \in X, f \in \mathcal{H}$. Folglich gilt $\langle k(\cdot, x), k(\cdot, x') \rangle = k(x, x')$. Ein Skalarproduktraum $\mathcal{H}$ wird reproduzierender Hilbertraum (RKHS) gennant, wenn $\mathcal{H}$ die reproduzierende Eigenschaft besitzt.
Werden Kernelmethoden verwendet so wird die in 3.5 definierte Risk Funktion mit

$$h^* = arg \min_{h_\ell \in H_{RKHS}} R_{emp}(h_\ell) + \lambda R(h_\ell) \tag{5.4}$$

in einem RKHS eingebettet, wobei H_{RKHS} der zur Kernelmethode korrespondierender reproduzierender Kernel Hilbertraum ist. Durch die Anwendung des Representer-Theorems lässt sich die Lösung von 5.4 als

$$h^* = \sum_{i=1}^{n} \alpha_i k(\cdot, x_i) \tag{5.5}$$

darstellen.

5.4 Kernelbasiertes SVM

Damit SVM mit Kernelmethoden erweitert werden kann, wird die duale Form des SVM Problems gebildet. Durch Verwendung des Lagrange-Multiplikators und Ausnutzung der

Karush-Kuhn-Tucker-Bedingungen lässt sich 4.5 zum äquivalenten dualen Problem umformen:

$$\begin{aligned} W(\alpha) &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i}^{n} \sum_{j}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle \\ \text{Unter der Nebenbedingungen: } & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq \frac{1}{2n\lambda} \text{für alle } i \in \{1, ..., n\}. \end{aligned} \tag{5.6}$$

Ausgehend vom dualen Problem 5.6 wird SVM um Kernelmethoden erweitert, indem $\langle x_i, xj \rangle$ durch $K(x_i, x_j)$ ersetzt wird.

$$\begin{aligned} W(\alpha) &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i}^{n} \sum_{j}^{n} y_i y_j \alpha_i \alpha_j \langle \phi(x_i), \phi(x_j) \rangle \\ \text{Unter der Nebenbedingungen: } & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq \frac{1}{2n\lambda} \text{für alle } i \in \{1, ..., n\}. \end{aligned} \tag{5.7}$$

Der Vektor w lässt sich durch Linearkombinationen als

$$w = \sum_{i=1}^{n} \alpha_i y_i \phi(x_i) \tag{5.8}$$

darstellen.
Da nur das Skalarprodukt $\langle \phi(x_i), \phi(x_j) \rangle$ berechnet wird, gehört SVM zu den Algorithmen, die den Kernel-Trick verwenden. Daher eignet sich SVM für die Nutzung von Kernelmethoden. Ein zu klassifizierendes Beispiel x wird analog zur in 4.7 definierten Entscheidungsfunktion vorhergesagt, wobei für w 5.8 und $x = \phi(x)$ gilt.

5.5 Kernelmethoden für Mustererkennung auf Bilddaten

Nach dem die Support Vector Machines mittels Kernelmethoden für den nicht linearen Gebrauch erweitert wurden, werden im Folgendem die in dieser Arbeit verwendeten Kernelmethoden vorgestellt.

5.5.1 Radial Basis Function Kernel

Die Radial Basis Function-Kernel (RBF-Kernel) ist im Kontext von SVM eine sehr populäre Kernelmethode.

Definition. Seien $x, x' \in X$ Datenvektoren, dann ist die RBF-Kernel Funktion definiert durch:

$$K(x, x') = \exp(-\gamma \|x - x'\|^2). \tag{5.9}$$

Der RBF-Kernel korrespondiert zu einem unendlich dimensionalen Skalarproduktraum, daher ist die Ausnutzung des Kernel-Tricks essentiell. Das für Kernelmethoden charakteristische Bewerten von Ähnlichkeit zwischen Datenvektoren x und x' wird durch die quadrierte Euklidische Distanz $\|x - x'\|^2$ deutlich. Angenommen x und x' seien identisch, das heißt: $x = x'$. Dann gilt für $K(x, x') = \exp(-\gamma \cdot 0) = \exp(0) = 1$. Unterscheiden sich x und x', dann ist $\|x - x'\|^2 > 0$ und damit ist $K(x, x') < 1$. Je stärker sich zwei Datenvektoren unterscheiden, desto kleiner wird der Wert der RBF-Funktion. Somit stellt der RBF-Kernel ein Ähnlichkeitsmaß dar.
Durch den γ-Parameter wird der Einflussbereich der einzelnen Trainingsbeispielen bestimmt. Eine hoher γ-Wert entspricht einen kleineren Einflussbereich der Datenbeispiele. Dies hat zur Folge, dass das Modell komplexer wird und somit die Gefahr der Überanpassung steigt. Analog gilt für kleine γ-Werte, dass der Einflussbereich der Beispiele steigt, wodurch die Modellkomplexität verringert wird.
Der RBF-Kernel ist eine populäre Kernelmethode, wenn man Mustererkennung auf Bilddaten betreiben möchte. Unter anderem wurde der Kernel erfolgreich in den Anwendungsgebieten der Gesichtserkennung [22] und Handschrifterkennung angewendet. Der MNIST Datensatz wurde unter Verwenden von SVM mit dem RBF-Kernel erfolgreich getestet. [19] [18] Daher müsste sich der RBF-Kernel für das Lernen auf FACT Daten grundsätzlich eignen.

5.5.2 Arccos-Kernel

Die im Folgendem betrachtete Kernelmethode stammt aus der Arccos-Kernel Familie [11], welche durch

$$k_n(\mathbf{x}, \mathbf{y}) = 2 \int \mathbf{dw} \frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{(2\pi)^{(\frac{d}{2})}} \Theta(\mathbf{w} \cdot \mathbf{x}) \Theta(\mathbf{w} \cdot \mathbf{y}) (\mathbf{w} \cdot \mathbf{x})^n (\mathbf{w} \cdot \mathbf{y})^n \tag{5.10}$$

für den Grad $n \in \mathbb{N}_0$ repräsentiert werden, wobei Θ die Heaviside Funktion mit $\Theta(z) = \frac{1}{2}(1 + sign(z))$ ist. Eine einfache Evaluation der in 5.10 gegebene Integral Repräsentation wird mit der Berechnung von

$$k_n(\mathbf{x}, \mathbf{y}) = \frac{1}{\pi} \|\mathbf{x}\|^n \|\mathbf{y}\|^n J_n(\theta) \tag{5.11}$$

ermöglicht, wobei

$$\theta = \arccos\left(\frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}\right) \tag{5.12}$$

und

$$J_n(\theta) = (-1)^n (\sin\theta)^{2n+1} \left(\frac{1}{\sin\theta} \frac{\partial}{\partial\theta}\right)^n \left(\frac{\pi - \theta}{\sin\theta}\right) \tag{5.13}$$

ist.
In dieser Arbeit wird die Arccos-Kernelmethode für den Grad eins verwendet. Folglich ist für $n = 1$

$$k_1(\mathbf{x}, \mathbf{y}) = \frac{1}{\pi} \|\mathbf{x}\| \|\mathbf{y}\| J_1(\theta) \tag{5.14}$$

mit

$$J_1(\theta) = \sin\theta + (\pi - \theta)\cos\theta \tag{5.15}$$

definiert. Die Arccos-Kernelmethoden besitzen Verbindungen zu Single Layer Threshold Networks [21]. Ein Single Layer Threshold Network stellt ein künstliches neuronales Netz mit einem Layer dar. Ein Single Layer Neuronales Netz besteht aus einer Eingabe $\mathbf{x} = x_1,...,x_d$ (Input) und Ausgabeelemente $\mathbf{f} = f_1,...,f_m$. Bezogen auf die FACT und MNIST Anwendungen würde jeder Input Knoten $x_i \in \mathbf{x}$ genau ein Pixel repräsentieren. Die Input Elemente werden mit den Einheiten der Ausgabe (=Klassen) verbunden. Die Verknüpfung zwischen Input Layer und Ausgabeeinheiten ist gewichtet und wird mittels einer Matrix $\mathbf{W}$ realisiert. Der Input $\mathbf{x}$ wird auf die Ausgabe $\mathbf{f}(\mathbf{x})$ mit $\mathbf{f}(\mathbf{x}) = g(\mathbf{W}\mathbf{x})$ abgebildet. Die Funktion g ist eine Aktivierungsfunktion, welche abhängig vom Grad n durch $g_n(z) = \Theta(z)z^n$ definiert ist und einen Werte zwischen 0 und 1 zurückgibt. Für $n = 1$ ist $g_1(z) = \Theta(z)z = \frac{1}{2}(1 + sign(z))$. Dies entspricht der ReLu Aktivierungsfunktion.
Die Verbindung zwischen 5.10 und einem Single Layer Threshold Network ergibt sich aus der Betrachtung von sehr großen Single Layer Threshold Networks. Das Skalarprodukt zwischen $f(\mathbf{x})$ und $f(\mathbf{y})$ wird durch

$$\langle \mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{y}) \rangle = \sum_{i=1}^{m} \Theta(\mathbf{w_i} \cdot \mathbf{x})\Theta(\mathbf{w_i} \cdot \mathbf{y})(\mathbf{w_i} \cdot \mathbf{x})^n(\mathbf{w_i} \cdot \mathbf{y})^n \tag{5.16}$$

definiert. Dann lässt sich für m gegen unendlich zeigen, dass 5.10 als Skalarprodukt für unendlich große Single Layer Threshold Networks 5.16 betrachtet werden kann [11].
Die Arccos-Kernelmethoden besitzen keine zusätzlichen kontinuierlichen Parameter, wie der γ-Parameter beim RBF-Kernel. Dadurch wird aufwändiges Parameter Tuning reduziert. Da Künstliche Neuronale Netze gute Ergebnisse bei der Mustererkennung auf Bilddaten aufzeigen [18], müsste sich die Arccos-Kernelmethode, aufgrund der Verbindung zu Single Layer Threshold Networks, für die Klassifikation der FACT-Daten eigen.

Kapitel 6

Effizientes kernelbasiertes Lernen

Kernelmethoden erweitern die Funktionalität von linearen Methoden auf nicht linear-trennbaren Daten. Durch Algorithmen, wie SVM, wird der Kernel-Trick genutzt, so dass Datenbeispiele in hochdimensionalen Räumen effizient berechnet werden können. Durch das Ausnutzen des Kernel-Tricks hängt der Aufwand eine Kernelmethode zu evaluieren nicht von der Dimensionalität des abgebildeten Raumes ab. Allerdings hängt die Effizienz kernelbasierten Lernens stark von der Eingabegröße ab. Besteht die Anforderung auf sehr vielen Daten zu trainieren, wie es bei den FACT-Versuchen der Fall ist, ist dies nicht effizient mit Kernel-SVM möglich. Die schlechte Skalierbarkeit von Kernel-SVM ist vordergründig durch die Größe der Kernel-Matrix begründet. Aus 5.2 wird ersichtlich, dass die Kernel-Matrix quadratisch zur Eingabegröße ist. Dies hat zur Folge, dass abhängig von der Eingabegröße die Kernel-Matrix Größen erreicht, welche durch Computersysteme nicht mehr berechnet werden können.
Soll mit Kernel-SVM auf eine Trainingsmenge S mit n Beispielen gelernt werden, dann wird für die Lösung von 5.7 eine Kernel-Matrix K der Größe $n \times n$ benötigt, was einen Speicherbedarf erfordert der in $\mathcal{O}(n^2)$ liegt. Des Weiteren wird zum Lösen der quadratischen Optimierungsaufgabe 5.7 Operationen auf K angewendet, die typischerweise in $\mathcal{O}(n^3)$ liegen. Durch die schlechte Skalierbarkeit auf großen Daten, können die FACT-Versuche und vergleichbare Anwendungen, nicht effizient durch Kernel-SVM gelernt werden. Daher ist das effiziente Lernen mit Kernelmethoden ein relevantes Forschungsgebiet.

6.1 Nyström Approximation

Die resultierende Größe der Kernel-Matrix stellt die größte Herausforderung beim effizienten Lernen mit Kernelmethoden auf großen Datensätzen dar. Ein naheliegender Ansatz, um Kernel-SVM skalierbar zu gestalten, ist die Approximation der Kernel-Matrix. Durch die Verwendung von approximierten Kernel-Matrizen lassen sich die notwendigen Ressour-

cen erheblich reduzieren. Die populärste Methode Kernel-Matrizen zu approximieren ist die Anwendung Nyström Approximation. [29]

Definition. Eine $n \times n$ Kernel-Matrix K wird gemäß der Nyström Approximation durch

$$\tilde{K} = K_{nm} K_{mm}^{-1} K_{mn} \tag{6.1}$$

approximiert.

Die zu approximierende Kernel-Matrix K wird in Blöcke der Größen $m \times m$, $n \times m$ und $m \times n$ partitioniert. Für die Berechnung von $\tilde{K}$ muss nur $K_{m,m}$ und $K_{n,m}$ explizit berechnet werden. Der Block $K_{m,n}$ ergibt sich aus der Bildung der Transponierten von $K_{n,m}$ und muss somit nicht explizit berechnet werden. Insgesamt ergibt sich einen Speicherbedarf von $\mathcal{O}(nm)$. Neben der Matrizen Multiplikation wird in 6.1 das Inverse von $K_{m,m}$ gebildet. Die Inversenbildung liegt für eine $m \times m$ Matrix in $\mathcal{O}(m^3)$. Entsprechend beträgt der gesamte Aufwand $\mathcal{O}(nm + m^3) = \mathcal{O}(nm^2)$. [29]
Offensichtlich beeinflusst die Größe von m den Aufwand zur Berechnung von $\tilde{K}$. Die Wahl von m bestimmt die Größe von $K_{n,m}$, $K_{m,n}$ und $K_{m,m}$ und damit die Anzahl der explizit zu berechnenden Kerneleinträge, was sich sowohl in benötigen Speicher als auch Laufzeit auswirkt. Neben dem Aufwand $\tilde{K}$ zu berechnen beeinflusst die Größe von m die Genauigkeit von $\tilde{K}$ bezüglich zu K.

6.1.1 Theorem. *Sei K eine Kernel-Matrix mit $Rang(K) = k$. Dann wird K durch* 6.1 *exakt approximiert, falls $m = k$ gilt und m linear unabhängige Zeilen/Spalten gewählt wurden.*

Wird m deutlich kleiner wie der Rang von K gesetzt, so wird K nicht exakt von $\tilde{K}$ approximiert. Über die Wahl von m wird der Zielkonflikt, zwischen einer akkuraten Approximation von K und den Aufwand $\tilde{K}$ zu berechnen, gesteuert. Dabei hängt der Aufwand quadratisch von m ab. Kleine Werte für m resultieren in einem geringeren Aufwand, allerdings weicht $\tilde{K}$ stärker von K ab. Ein Maß für Abweichungen zwischen Matrizen liefert die Betrachtung der Frobenius Norm.

Definition. Für eine Matrix $A \in \mathbb{R}^{m \times n}$ wird die Frobenius Norm definiert durch: $\|A\|_F := \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$.

Die Abweichungen zweier Matrizen $A, B \in \mathbb{R}^{m \times n}$ lassen sich durch

$$\|A - B\|_F \tag{6.2}$$

quantifizieren.
Abgesehen von der gewählten Größe von m und den gegebenen Rang von K, hängt die

Genauigkeit von $\tilde{K}$ durch die Auswahl der m Spalten/Zeilen ab. Für die Approximation geeignete Spalten/Zeilen müssen linear unabhängig sein. Linear abhängige Spalten/Zeilen tragen keine neuen Informationen über die Struktur der Matrix und sind daher für die Approximation redundant.
Unter dem Begriff *Landmark Selection* existieren mehrere Strategien, die versuchen geeignete Spalten/Zeilen zu finden. Eine einfache und populäre Auswahlstrategie ist die Ziehung von m Spalten/Zeilen, die gleichverteilt aus K gezogen werden. Dieser Ansatz basiert auf der Generierung von Zufallszahlen, die die Indizes der selektierten Spalten/Zeilen repräsentieren. Das Ziehen einer großen Anzahl an Zufallszahlen ist sehr effizient möglich. Folglich ist die zufällige Auswahl der Indizes auch für große Datensätze gut skalierbar. Neben der gleichverteilten Ziehung der Indizes, existieren Auswahlstrategien, die auf anderen Wahrscheinlichkeitsverteilungen beruhen, die versuchen den Approximationsfehler weiter zu senken [12].
Eine andere Klasse von Selection-Strategien beruhen auf dem K-means Algorithmus. Häufig sind die Approximation, deren Spalten/Zeilen durch K-means selektiert wurden, akkurater als bei der zufälligen Auswahlstrategie [17]. Allerdings skaliert der K-means Algorithmus schlecht für große Matrizen, weshalb der Einsatz für große Eingaben problematisch ist.

6.2 Effizientes linearisieren von Modellen

Durch den Einsatz der Nyström Approximation lässt sich der Aufwand der in 5.7 definierten Kernel-SVM Aufgabe von $\mathcal{O}(n^3)$ zu $\mathcal{O}(nm^2)$ reduzieren. Abhängig von der gewählten Größe des Parameters m, wird das Lernen mit Kernelmethoden durch die Nyström Approximation beschleunigt und die Ressourcenanforderungen verringert. Entsprechend ist die Approximation der Kernel-Matrix ein wichtiges Werkzeug, um Kernel-SVM auf vielen Daten skalierbar zu gestalten.
Zwar verringert der Einsatz der Nyström Approximation die Laufzeit und den Speicherbedarf, welcher bei der Speicherung und Manipulation der Matrix entsteht, allerdings stellt 5.7 eine Aufgabe der quadratischen Programmierung dar. Entsprechend kann zum Lösen von 5.7 nicht auf die sehr effizienten und speicherschonenden linearen Methoden zurückgegriffen werden. Typischerweise besitzen lineare Klassifikatoren einen linearen Aufwand bezüglich der Eingabegröße und skalieren somit besser auf großen Datensätzen als quadratische Lösungsmethoden. Daher existiert das Bestreben Kernel-SVM zu linearisieren. Das Ziel der Linearisierung ist es, die quadratische Programmierungsaufgabe in einer äquivalenten linearen Problemformulierung umzuwandeln, welche durch lineare Methoden gelöst werden kann. Der Grundlegende Ansatz der Linearisierung beruht auf die Dekomposition der Kernel-Matrix.
Sei $K_{TRAIN} \in \mathbb{R}^{n \times n}$ eine Kernel-Matrix für n Trainingsdaten. Da K_{TRAIN} eine gültige Kernel-Matrix ist, muss sie symmetrisch und positiv semidefinit sein. Dann existiert

von K_{TRAIN} eine Eigendekomposition mit $K_{TRAIN} = U\Lambda U^T$. Die Trainingsdaten werden modelliert zu $X_{train_{LINEAR}} = U\Lambda^{\frac{1}{2}}$. Anschließend wird $X_{train_{LINEAR}}$ durch lineares SVM gelernt. Es entsteht ein lineares Modell SVM_{linear}.
Sei $K_{TEST} \in \mathbb{R}^{l \times n}$ eine Kernel-Matrix für l Testdaten. Dann werden die Beispiele durch $y = w \cdot X_{test_{LINEAR}}$ vorhergesagt, wobei $X_{test_{LINEAR}} = U\Lambda^{-\frac{1}{2}}$ ist. Ein solches Vorgehen ermöglicht die Generierung von linearen Modellen, die äquivalent zu 5.7 sind.
Die beschriebene Umwandlung in ein lineares Modell erfordert die Dekomposition der Kernel-Matrix. Da die Eigendekomposition eine Operation ist, welche in $\mathcal{O}(n^3)$ liegt, ist die dargestellte Linearisierung für große Datensätze nicht skalierbar und somit für effizientes Lernen auf großen Datensätzen unbrauchbar.
Damit die Linearisierung für große Eingabegrößen gelingt, müssen die Kosten für die Berechnung der Eigendekomposition der Kernel-Matrix reduziert werden. Wiederrum stellt die Approximation ein Ansatz dar, um die entstehen Kosten der Linearisierung zu begrenzen. Für die Approximation der Eigendekomposition lässt sich die eingeführte Nyström Approximation verwenden.
Der Trainingskernel wird mit

$$\tilde{K}_{TRAIN} = K_{nm} K_{mm}^{-1} K_{mn} \tag{6.3}$$

durch Nyström approximiert. Unter der Anwendung der Nyström Approximation folgen für die linearisierten Trainings- und Testdaten:

$$\tilde{X}_{train_{LINEAR}} = K_{nm} U_m \Lambda_m^{-\frac{1}{2}} \tag{6.4}$$

und

$$\tilde{X}_{test_{LINEAR}} = K_{TEST} U_m \Lambda_m^{-\frac{1}{2}} \tag{6.5}$$

Da $\tilde{X}_{train_{LINEAR}} \in \mathbb{R}^{n \times m}$ und $\tilde{X}_{test_{LINEAR}} \in \mathbb{R}^{l \times m}$ ist, beeinflusst wiederum der Parameter m, den für die Linearisierung benötigten Aufwand.
Die Linearisierung unter Anwendung der Nyström Approximation wird durch den *low-rank linearized SVM* (LLSVM) Algorithmus in 6.1 realisiert. [30] In Abhängigkeit von m, ermöglicht der LLSVM Algorithmus eine lineare Laufzeitkomplexität, sowie einen konstanten Speicherbedarf bezüglich der Eingabegröße. Das Vorhersagen eines Datenbeispiels ist mit $\mathcal{O}(c_1 m^2 + c_2 m)$ ebenfalls abhängig von der Größe von m [30]. Durch die Approximation der Eigendekomposition wird die Linearisierung des Modells auch für großen Eingabegrößen möglich.

6.3 Stochastic Gradient Descent

Durch Anwendung des LLSVM-Algorithmus ist es möglich, effiziente lineare Methoden zum Lösen zu verwenden. Eine beliebter linearer Klassifikator ist die Stochastic Gradient

1: **procedure** TRAININGSPHASE(X_{train}, y_{train}, m)
2: $z \leftarrow$ Ziehe m Spalten/Zeilen aus X_{train}
3: Berechne K_{zz} und $U_z \Lambda_z U_z^T$
4: Berechne $M = U_z \Lambda_z^{-1/2}$
5: Berechne K_{nz}
6: $\tilde{X}_{train_{LINEAR}} = K_{nz} M$
7: $SVM_{LINEAR} \leftarrow$ Trainiere lineare SVM auf $\tilde{X}_{train_{LINEAR}}$
8: **end procedure**
9: **procedure** TESTPHASE(M,X_{test})
10: Berechne Testkernel-Matrix K_{lz}
11: Berechne $\tilde{X}_{test_{LINEAR}} \leftarrow K_{lz} M$
12: Vorhersage durch $SVM_{LINEAR}(\tilde{X}_{test_{LINEAR}})$
13: **end procedure**

Algorithmus 6.1: LLSVM Algorithmus

Descent Methode (SGD). Grundlage der SGD Methode ist die schrittweise Verbesserung eines Modells durch die Gradientenbildung der Loss Funktion. Im Gegensatz zur Gradient Descent Methode wird beim Stochastic Gradient Descent in jedem Schritt nur der Gradient von einem Trainingsbeispiel berechnet. Durch diese Approximation des Gradienten ist die SGD-Methode effizienter als die Gradient Descent Methode.

Ablauf: SGD-Methode

1. Initialisiere w und wähle die Schrittweite η
2. Solange Abbruchkriterium nicht erreicht:
   (a) Zufälliges mischen der Trainingsdaten.
   (b) $w := w - \eta(\alpha \frac{\partial R(w)}{\partial w}) + \frac{\partial L(w^T x_i + b, y_i)}{\partial w}$ (Update Regel)

Nach der Initialisierung iteriert SGD über alle Trainingsdaten, wobei für jede Iteration die Update Regel angewendet wird. Nach jeder Iteration wird geprüft, ob das Abbruchkriterium erfüllt wurde. Falls es gilt, wird w zurückgegeben und SGD terminiert. Wurde über jedes Trainingsbeispiel iteriert, wird der Trainingsdatensatz durchmischt und es startet ein neuer Durchlauf.
Bei der Anwendung der Update Regel wird der Parameter w gemäß des gebildeten Gradienten optimiert. Dabei wird der Gradienten über die Loss Funktion L, sowie des Regularisierungsausdrucks R gebildet. Der Parameter α ist der Regularisationsparameter, der die Komplexität von w gemäß seiner Gewichtung bestraft. Wie stark sich in jeder Iteration w durch die Update Regel verändert, wird durch die Schrittweite η bestimmt. Entsprechend

wird w für größere η stärker durch die Update Regel verändert. Eine häufig gewählte Strategie ist es, mit fortschreitenden Iterationen η zu verringern. Das Abbruchkriterium kann durch eine konstante Anzahl an Durchläufen über die Trainingsdaten gegeben sein (Epochen) oder durch die Definition eines Toleranzwertes.
Durch die Approximation des realen Gradienten ist das SGD Verfahren eine sehr effiziente Lösungsmethode für lineare Probleme. Vor allem auf großen Datensätzen ist der Einsatz von SGD prädestiniert, da die Komplexität meistens linear zur Eingabegröße bleibt. [31] [4]
Allerdings ist das Klassifizieren mit SGD häufig mit aufwändiger Parameteroptimierung verbunden, da insbesondere geeignete Werte für α gefunden werden müssen. Zusätzlich muss das Konvergenzverhalten von SGD abgeschätzt werden, um geeignete Abbruchkriterien zu finden.
Aufgrund der Gradientenbildung über der Loss Funktion, setzt SGD voraus, dass die verwendete Loss Funktion differenzierbar ist. Bezogen auf das Lösen von SVM entsteht das Problem, dass die Hinge Loss Funktion nicht differenzierbar ist. Eine Gradientenbildung von L_{HINGE} ist somit nicht möglich. Jedoch existiert für L_{HINGE} ein Subgradient, der das Lösen von linearen Support Vector Machines durch SGD ermöglicht.

Kapitel 7

Experimentieren mit Modellen

Durch die Linearisierung von Kernel-SVM unter Nutzung der Nyström Approximation, ist das notwendige Rüstzeug vorhanden, um effizientes kernelbasiertes Lernen auf vielen Daten zu ermöglichen. Damit sind die Voraussetzungen gegeben, linearisierte Modelle auf Grundlage von vielen, nicht linear-trennbaren, Trainingsdaten zu generieren.
Es stellt sich die Frage, wie geeignet die Modelle für Vorhersagen von Datenbeispiele sind, und unter welchem Aufwand sie generiert werden können. Diese Fragen werden durch systematisches Testen der erzeugten Modelle in Form einer Evaluation beantwortet. Bevor die Leistungsfähigkeit der verwendeten Methoden eingeschätzt werden kann, müssen einige notwendige Konzepte eingeführt werden.

7.1 Metrik

In der experimentellen Evaluation werden Aussagen auf Grundlage von Versuchen getätigt. Das bedeutet, es wird systematisch mit Modellen experimentiert, mit dem Ziel die Leistungsfähigkeit der Modelle einzuschätzen. Damit die Performanz eines Modells bewertet werden kann, wird eine Metrik benötigt.
Eine Metrik quantifiziert den auftretenden Testfehler eines Vorhersagemodells. Voraussetzung den Testfehler ermitteln zu können, sind Daten mit bekannter Klassifikation $y \in \{-1, 1\}$.

Definition. Sei $\hat{\mathcal{Y}}$ die prognostizierten Klassenzugehörigkeiten und $\mathcal{Y}$ die tatsächlichen Klassen von Datenbeispielen.
Eine Metrik ist eine Funktion $M : \hat{\mathcal{Y}} \times \mathcal{Y} \longmapsto \mathbb{R}$, die die Korrektheit der Vorhersagen bewertet.

Ausgehend von einer binären Klassifikation und gegebenen $\mathcal{Y}$ ist die Anzahl positiver (p) und negativer (n) Beispiele bekannt. Des Weiteren lässt sich durch $\mathcal{Y}$ und $\hat{\mathcal{Y}}$ definieren:

True-Positive (TP): Anzahl der positiven Beispiele, die korrekt vorhergesagt wurden.

True-Negative (TN): Anzahl der negativen Beispiele, die korrekt vorhergesagt wurden.

False-Positive (FP): Anzahl der negativen Beispiele, die inkorrekt vorhergesagt wurden.

False-Negative (FN): Anzahl der positiven Beispiele, die inkorrekt vorhergesagt wurden.

True-Positive-Rate (TPR): $TPR = \frac{TP}{p}$

False-Positive-Rate (TPR): $TPR = \frac{FP}{n}$

Die durch ein Modell korrekt vorhergesagten Daten werden durch TP und TN ausgedrückt. Entsprechend sind FP und FN die Anzahl der falsch klassifizierten Beispiele.
Eine häufig verwendete Metrik ist die Accuracy,

$$ACC = \frac{TP + TN}{p + n} \tag{7.1}$$

die das Verhältnis aus der Summe der korrekt klassifizierten Objekten, mit der Gesamtanzahl der getesteten Objekte bildet.
Für einen Testdatensatz der die Klassen durch ungefähr gleich viele Datenbeispiele repräsentiert, bildet die Accuracy (ACC) häufig die Leistungsfähigkeit eines Klassifikators hinreichend genau ab. Wird auf einem nicht ausbalancierten Datensatz getestet, dann ist es möglich, dass die Accuracy die Leistungsfähigkeit eines Modells nur sehr verzerrt widerspiegelt. Dies lässt sich anhand eines Beispiels verdeutlichen:
Angenommen ein Modell würde jedes Beispiel als positives Beispiel klassifizieren. Weiter sei ein Testdatensatz von 10000 Beispielen gegeben, welcher aus 8000 positiven (p=8000) und 2000 (n=2000) negativen Beispielen besteht. Dann wäre die Accuracy gegeben durch $ACC = \frac{8000}{10000} = 0.8$.
Da das Modell im Beispiel keine negativen Beispiele erkennen kann, besitzt es hinsichtlich einer Klassifikation keinen Nutzen. Trotzdem wird das nutzlose Modell mit einer Accuracy von 80% bewertet. Offensichtlich spiegelt die Accuracy die wirkliche Klassifikationsperformanz des Modells nicht wieder.
Eine Möglichkeit die Performanz eines Modells für das beschriebene Szenario adäquater einzuschätzen, stellt die Betrachtung der ROC-Kurve dar.
Die ROC-Kurve (Receiver Operating Characteristic) wird zwischen der True-Positive-Rate und der False-Positive-Rate aufgespannt. Im Kontext dieser Arbeit wird der AUC-Wert ermittelt, der die Fläche unterhalb der konstruierten Kurve widerspiegelt. Der AUC-Wert kann zwischen 0 und 1 liegen, wobei die erreichte Performanz mit $AUC = 0.5$ am niedrigsten bewertet wird. Bezogen auf dem Klassifikator aus dem Beispiel, würde ein AUC Wert von 0.5 ermittelt werden, was seiner tatsächlichen Leistungsfähigkeit entspräche.

Neben der Wahl einer geeigneten Metrik ist das Wiederholen von Experimenten wichtig, um einen adäquaten Einblick in die Leistungsfähigkeit der Modelle und somit der verwendeten Methoden zu bekommen. Dies gilt vor allem, wenn Methoden verwendet werden, welche stochastische Elemente besitzen. Bezogen auf die durchzuführenden Experimente, obliegt die Nyström Approximation einen Zufallseinfluss durch die randomisierte Wahl der Spalten/Zeilen. Außerdem weißt der SGD-Klassifikator einen Zufallseinfluss auf, der durch das Durchmischen der Trainingsmenge entsteht. Es ist demnach davon auszugehen, dass die ermittelten Ergebnisse schwanken. Daher wird in dieser Arbeit jeder Versuch mehrfach durchgeführt, wobei die Ergebnisse durch Durchschnittswerte und den jeweiligen Standardabweichungen gegeben sind.

7.2 Methoden der Modelloptimierung

Die Verwendung von geeigneten Metriken ermöglichen es, die Performanz von Modellen zu bewerten. Das Ziel beim Klassifizieren von Daten ist es durch Klassifikatoren Modelle zu erzeugen, die einen möglichst geringen Testfehler aufweisen. Dabei existieren einige Freiheitsgrade bei der Modellgewinnung, welche die Performanz eines Vorhersagemodells beeinflussen können.

7.2.1 Modellanpassung durch Parameteroptimierung

Typischerweise wird durch gezielte Modellanpassungen versucht den Testfehler zu minimieren. Dies wird durch das Optimieren von Modellparameter erreicht. Wie bereits dargelegt, bietet das Minimieren der regularisierten empirischen Risk Funktion einen Regularisationsausdruck. Daher bietet es sich häufig an, λ und R zu optimieren. Beispielsweise ist es empfehlenswert, SVM unter Verwendung des RBF-Kernels, geeignete Parameter für den Regularisationsparameter und γ zu finden [15]. Eine Möglichkeit geeignete Parameter zu finden, bietet die Grid-Search Methode. Dabei werden systematisch definierte Parameterkombinationen getestet und anhand einer Metrik bewertet. Damit es bei der Modellanpassung nicht zu einer Überanpassung des Modells kommt, wird häufig die Performanz durch Cross-Validation ermittelt. (Grid-SearchCV)

7.2.2 Repräsentation von Datenvektoren

Neben der Modellanpassung durch Parametertuning, kann die Repräsentation des Datenvektors Einfluss auf die Performanz des Klassifikators ausüben.

Skalierung

Häufig werden Skalierungen verwendet, die die Werte eines Datenvektors in einem definierten Intervall skalieren. Durch die Skalierung der Werte in beispielsweise $[0, 1]$ oder $[-1, 1]$ können große Differenzen zwischen den Werten verringert werden. Dies verhindert, dass sehr große Werte zu dominant werden und beugt nummerische Instabilitäten vor. Die Skalierung von Trainings- und Testdaten, wird durch einen Skalierungsfaktor berechnet. Dieser ist entweder a prioi bekannt, oder muss aus den Trainingsdaten ermittelt werden. Beispielsweise befindet sich jeder Wert eines Datenvektors aus dem MNIST Datensatzes zwischen $[0, 255]$. Folglich ist der Skalierungsfaktor $\frac{1}{255}$. Für die FACT-Versuche sind die maximalen Werte (= maximale Anzahl auftreffender Photonen) nicht bekannt. Daher muss der Skalierungsfaktor aus dem Trainingsdatensatz ermittelt werden. Die Skalierung, dessen Skalierungsfaktor aus dem maximalen Wert berechnet wurde, wird im Folgendem $MaxAbs$-Skalierung genannt.

Normalisierung

Bei der Normalisierung wird jeder Datenvektor anhand seiner Euklidischen Norm skaliert. Ein Datenvektor x wird durch

$$x_{norm} = \frac{1}{\|x\|} \tag{7.2}$$

normiert.
Die in 7.2 definierte Norm entspricht der L_2 Norm mit $\|x\| = \sqrt{x_1^2, ..., x_n^2}$.

Standardisierung

Das Standardisieren eines Datenvektors x wird durch

$$x_{st} = \frac{x - mean(x)}{sd(x)} \tag{7.3}$$

realisiert, wobei $mean(x)$ das arithmetische Mittel und $sd(x)$ die Standardabweichung von x ist.
Standardisierte Datenvektoren entsprechen einer Normalverteilung $\mathcal{N}(\mu, \sigma^2)$ mit $\mu = 0$ und $\sigma^2 = 1$. Viele maschinelle Methoden wurden unter der Annahme konzipiert, dass die Daten gemäß der Verteilung $\mathcal{N}(0, 1)$ vorliegen.

Kapitel 8

Effizientes Lernen - MNIST Datensatz

Im Folgendem wird der LLSVM-Algorithmus auf dem MNIST Datensatz angewendet. Im Fokus der Untersuchungen stehen Modelle, die durch LLSVM erzeugt werden. Bei den Experimenten wird zum einen das erreichte Niveau der Performanz der Klassifikation, zum anderen der benötigte Aufwand zur Modellerzeugung analysiert. Insbesondere wird der Einfluss des Parameters m auf den benötigten Aufwand und der erreichten Performanz untersucht.

Versuchsaufbau

Der MNIST Datensatz ist gegeben mit 60000 Trainingsdaten und einer Testmenge von 10000 Beispielen. Sowohl die Trainings- als auch die Testmenge sind bezüglich ihrer Klassenvorkommen ausbalanciert. Bei den Experimenten wird die Accuracy 7.1 als einzige Metrik erhoben. Dies ist durch die ausbalancierte Testmenge und den bereits bekannten Ergebnisse zum MNIST Datensatz begründbar.
Getestet werden der RBF- und Arccos-Kernel unter Anwendung des LLSVM-Algorithmus, wobei als linearer Klassifikator die Stochastic Gradient Descent Methode verwendet wird. Jeder Versuch wird auf den kompletten Trainingsdatensatz gelernt und auf den vollständigen Testdaten getestet. Des Weiteren wurde jede Versuchsreihe mehrfach wiederholt, so dass Durchschnittswerte mit zugehöriger Standardabweichung erhoben wurden.
Neben der erreichten Performanz wird der benötigte Aufwand zur Modellerzeugung ermittelt und analysiert. Bei der Analyse des Aufwandes steht die benötigte Laufzeit für die Berechnung von 6.4 beziehungsweise 6.5 im Fokus der Untersuchung. Dabei wird die durchschnittliche Laufzeit zur Erstellung der transformierten Datenvektoren in R_{Linear} zusammengefasst. Von der Laufzeit des verwendeten SGD-Klassifikators wird zunächst abstrahiert.

Wahl der Datenrepräsentation

Die MNIST Experimente wurden mit unterschiedlichen Datenrepräsentationen getestet. Für den Arccos-Kernel erreichen die Original Repräsentation, MaxAbs-Skalierung, sowie die L2 normalisierten Daten das gleiche Performanz-Niveau. Die Standardisierung der Daten resultiert in einen ca 1% höheren Testfehler.
Gemessen an der Performanz, kommen für den RBF-Kernel nur eine nicht skalierte oder eine MaxAbs-Repräsentation der Daten in Frage.
Für die folgenden Versuchsreihen wurden für beide Kernelmethoden die MaxAbs-skalierten Daten verwendet.

		Original	MaxAbs	L2 Norm	Standardisiert
RBF-Kernel	ACC	98.32	98.37	91.92	91.46
	sd(ACC)	0.0345	0.0237	0.612	0.4827
Arccos-Kernel	ACC	98.32	98.34	98.3	97.32
	sd(ACC)	0.0379	0.0349	0.0379	0.0971

Tabelle 8.1: Datenrepräsentation - MNIST

Optimieren der Parameter

Geeignete Parameter für die Regularisierung (und beim RBF-Kernel für den γ-Parameter) wurden mit dem Grid-Search Verfahren über einen definierten Parameterraum gefunden:

- Arccos-Kernel: $\alpha = 1e^{-5}$, $R = L2$
- RBF-Kernel: $\alpha = 1e^{-5}$, $R = L2$, $\gamma = 0.01$.

Es wurde ebenfalls kurz mit unterschiedlichen Loss Funktionen experimentiert (Square Hinge, Perzeptron), was zu keiner höheren Performanz führte. Allerdings zeigt es, die hohe Flexibilität des SGD-Klassifikators, der nicht auf das Lösen von Support Vector Machines durch die Hinge-Loss Funktion beschränkt ist.

Anwendung LLSVM - Parameter m

Zunächst wird die erreichte Performanz der linearisierten und approximierten Modelle untersucht. In 8.2 ist die erreichte Performanz in Abhängigkeit von m für den RBF- und Arccos-Kernel gegeben.
Für $m = 5000$ wird mit 98.33% und 98.24% das jeweils niedrigste Accuracy-Niveau für den RBF beziehungsweise Arccos-Kernel erreicht. Die schlechtere Performanz deutet darauf hin, dass die Kernel-Matrix zu ungenau für $m = 5000$ approximiert wurde.

Ein stabiles Performanz-Niveau wird jeweils für $m >= 10000$ erreicht. In jedem untersuchten Fall, erreichte der LLSVM-Algorithmus unter Anwendung des RBF-Kernels ein durchschnittlich höheres Performanz-Niveau, als unter Verwendung des Arccos-Kernels. Das beste Ergebnis wird für $m = 20000$ mit 98.49% ACC durch den RBF-Kernel erreicht. Dieses Performanz-Niveau deckt sich mit den Vergleichsergebnisse für RBF-SVM auf den MNIST Datensatz. [18]

	RBF		Arccos	
m	ACC	sd	ACC	sd
5000	98.33	0.07	98.24	0.0416
10000	98.45	0.0289	98.38	0.0427
15000	98.44	0.05	98.34	0.0206
20000	98.49	0.0403	98.35	0.0507

Tabelle 8.2: Performanz: LLSVM - MNIST

Neben der erreichten Performanz beeinflusst die Größe von m den benötigten Aufwand zur Modellgenerierung. Wie zu erwarten war, wächst die für die Linearisierung benötigte Laufzeit quadratisch zu m. Für die MNIST Versuche scheint die Wahl von $m = 10000$ ein guter Kompromiss zwischen erreichter Performanz und benötigten Aufwand zu sein.

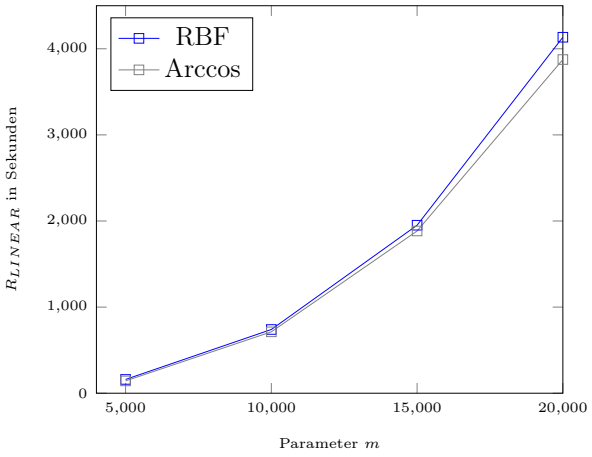


Abbildung 8.1: Laufzeit

Kapitel 9

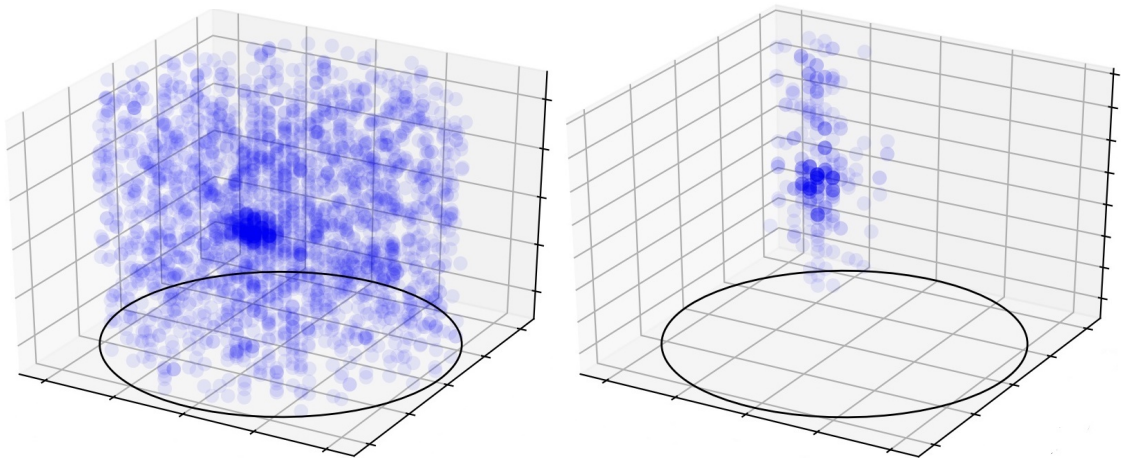
Effizientes Lernen - FACT Datensatz

Die Ergebnisse der MNIST Evaluation zeigen, dass unter Nutzung von Linearisierung unter der Verwendung der Nyström Approximation, Kernel-SVM auf dem kompletten MNIST Datensatz effizient anwendbar ist. Die Laufzeit und Ressourcenanforderungen waren durch die Setzung des Parameters m begrenzt. Falls m hinreichend groß gewählt wurde, konnte für das Setting erwartbare Performanz-Niveau erreicht werden. Im Folgendem wird mit der gleichen Methodik auf FACT Daten gelernt.
Erfolgreiches Lernen auf FACT-Daten setzen einen deutlich höheren Aufwand in der Datenaufbereitung voraus. In der Arbeit wird die Photon-Stream Repräsentation verwendet, die es ermöglicht, das Aufbereiten der FACT-Daten zu vereinfachen.

Photon-Stream Repräsentation

Wie bereits bekannt, werden Luftschauer durch die Photonensensibilität des FACT-Teleskops detektiert. In der Vergangenheit wurden die auftreffenden Photonen als einziger Signalimpuls interpretiert. Durch die Analyse des Signals wurden die Photonenäquivalenzen für jeden Pixel der FACT-Kamera abgeleitet. Auf diesem Weg wurden erfolgreich FACT-Experimente durchgeführt.
Die Photon-Stream Repräsentation ermöglicht die Darstellung von einzelnen Photonen. Dies wird durch die Rekonstruktion der Photonen aus dem Hauptsignalimpuls ermöglicht [5]. Durch die Repräsentation der einzelnen Photonen können Luftschauer akkurater und verständlicher dargestellt werden, da neben der Photonenhäufigkeiten auch die Ankunftszeiten der Phototonen rekonstruiert werden. Der Informationsvorteil lässt sich ebenfalls für die Datenaufbereitung nutzen.
Da Ort und Zeit jedes ankommenden Photons bekannt sind, lassen sich die Photonen in einem dreidimensionalen Koordinatensystem darstellen. Diese Darstellung als Punktwolke, visualisiert das Auftreffen von Photonen in jedem Pixel der Kamera. Der elementare Schritt der FACT Datenaufbereitung ist das Filtern von relevanten Photonen vom irre-

levanten Grundrauschen. Für das Training relevante Muster, weisen eine höhere Photonendichte auf. Daher besteht in der Punktwolken-Darstellung die Anforderung, Bereiche mit hoher Photonendichte zu identifizieren. Dies kann durch Clusterverfahren maschinell realisiert werden. Insbesondere bietet sich der Einsatz von DBSCAN an, einer Methode der Clusteranalyse, der Bereiche mit hoher Dichte identifiziert. Die Datenaufbereitung wurde in der Arbeit durch das Photon-Stream Package [1] realisiert, welches das DBSCAN-Clusterverfahren mit vordefinierten Parametern bietet.



(a) vor DBSCAN (b) nach DBSCAN

Abbildung 9.1: FACT - Datenaufbereitung

Datenrepräsentation

Nachdem DBSCAN die FACT-Daten vom störenden Grundrauschen befreit hat, werden die Daten in eine Bildrepräsentation überführt. Für jede FACT-Aufnahme entstand ein Datenvektor $x \in \mathbb{R}^{1440}$, der die Summe der aufgetroffenen Photonen in jedem Pixel widerspiegelt.
Der Datenvektor x kann wiederum in andere Repräsentationen überführt werden.

		Original	MaxAbs Skalierung	L2 Norm	Standardisiert
RBF-Kernel	ACC	64.64	51.92	56.9	63.4
	sd(ACC)	0.2411	0.4235	0.36	0.2722
Arccos-Kernel	ACC	64.94	61.88	67.27	66.1
	sd(ACC)	0.3287	1.1314	0.1758	0.4392

Tabelle 9.1: Datenrepräsentation - FACT
Getestet für $m = 10000$ auf 100000 Trainingsdaten

Für den Arccos-Kernel wird eine $L2$-Normalisierung verwendet, die die besten Ergebnisse lieferte. Beim RBF-Kernel wird auf eine Skalierung,Normalisierung oder Standardisierung verzichtet.

Separation Gamma/Proton Luftschauer

Der Versuchsaufbau ist analog zu den MNIST Versuchen mit dem Unterschied, dass als Metrik der AUC-Wert erhoben wurde. Geeignete Parameter wurden durch das Grid-SearchCV Verfahren gefunden:

- Arccos-Kernel: $\alpha = 1e^{-6}$, $Reg = L2$
- RBF-Kernel: $\alpha = 1e^{-6}$, $Reg = L2$, $\gamma = 0.001$.

Der SGD-Klassifikator wurde auf ausbalancierten Trainingsdatensätzen angewendet. Für jedes Experiment wurde die erreichte Performanz anhand von 30000 Testdaten ermittelt. Dabei war die Testmenge mit einer Verteilung von 10241 Proton-Daten und 19759 Gamma-Beispielen nicht ausbalanciert.

Performanz

Die erreichten AUC-Werte der Modelle wurden, in Abhängigkeit zu der gewählten Kernelmethode, der Größe des Trainingsdatensatzes, sowie des Parameters m erhoben. Dauerten Experimente länger als 20 Stunden, wurden sie abgebrochen (symbolisiert durch -). Der RBF-Kernel konnte nur weniger effizient evaluiert werden, daher wurde beim RBF-Kernel auf die Betrachtung von $m = 20000$ verzichtet.
Die Ergebnisse zeigen, dass das Trainieren von vielen Luftschauern zu akkurateren Vorhersagen führt. Die erreichte Performanz des RBF-Kernels ist in jedem untersuchten Fall signifikant schlechter, als beim Verwenden des Arccos-Kernels. Jeweils wurden die höchsten AUC-Werte für $m = 20000$ ermittelt. Für den kleinsten Datensatz, bestehend aus 60000 FACT-Beispielen, konvergiert die Performanz für den Arccos-Kernel bereits für $m = 15000$. Zusammenfassend wurde die beste Performanz mit 68.71% unter dem Arccos-Kernel für $m = 20000$ und dem Lernen von 170000 $L2$ normalisierten FACT-Daten erreicht.

$\|S\|$		5000	10000	15000
60000	AUC (%)	63.04	63.84	64.01
	sd(AUC)	0.0473	0.1258	0.245
	R_{LINEAR} (Min)	9	61	773
100000	AUC	63.61	64.51	64.93
	sd(AUC)	0.1273	0.099	0.163
	R_{LINEAR} (Min)	10	64	873
140000	AUC	64.18	65.23	65.71
	sd(AUC)	0.2165	0.1202	0.0707
	R_{LINEAR} (Min)	9	57	903
170000	AUC	64.21	65.21	-
	sd(AUC)	0.1202	0.0707	-
	R_{LINEAR} (Min)	11	74	-

Tabelle 9.2: LLSVM - RBF Kernel

$\|S\|$		5000	10000	15000	20000
60000	AUC	64.91	66.23	66.64	66.65
	sd(AUC)	0.3015	0.1011	0.0947	0.1511
	R_{LINEAR} (Min)	3	14	36	76
100000	AUC	65.88	67.45	67.72	67.79
	sd(AUC)	0.1757	0.1127	0.0966	0.2881
	R_{LINEAR} (Min)	4	17	42	82
140000	AUC	66.39	67.5	68.28	68.26
	sd(AUC)	0.2121	0.1485	0.0566	0.1131
	R_{LINEAR} (Min)	6	27	80	206
170000	AUC	66.65	67.77	68.53	68.71
	sd(AUC)	0.4031	0.2404	0.1768	0.0849
	R_{LINEAR} (Min)	8	38	153	381

Tabelle 9.3: LLSVM - Arccos Kernel

Aufwand

Analog zu den Ergebnissen des MNIST-Datensatzes, wächst der Aufwand quadratisch mit der Größe von m. Ebenfalls beeinflusst die Größe der Trainingsmenge den benötigten Aufwand.

Abschließend wird das Konvergenzverhalten des SGD-Klassifikators analysiert. Es zeigt

sich, dass die Konvergenz für $m = 20000$ am stabilsten Verläuft. Für $m < 20000$ treten tendenziell größere Schwankungen im Konvergenzverhalten auf. Häufig wird ein ausreichendes Konvergenz-Niveau für 300 oder mehr Durchläufen erreicht. Neben der Anzahl der durchgeführten Epochen, hängt die Laufzeit von der Größe von m ab. Dabei wird erkennbar, dass in Abhängigkeit zu m,die Laufzeitunterschiede mit steigenden Epochen größer werden.

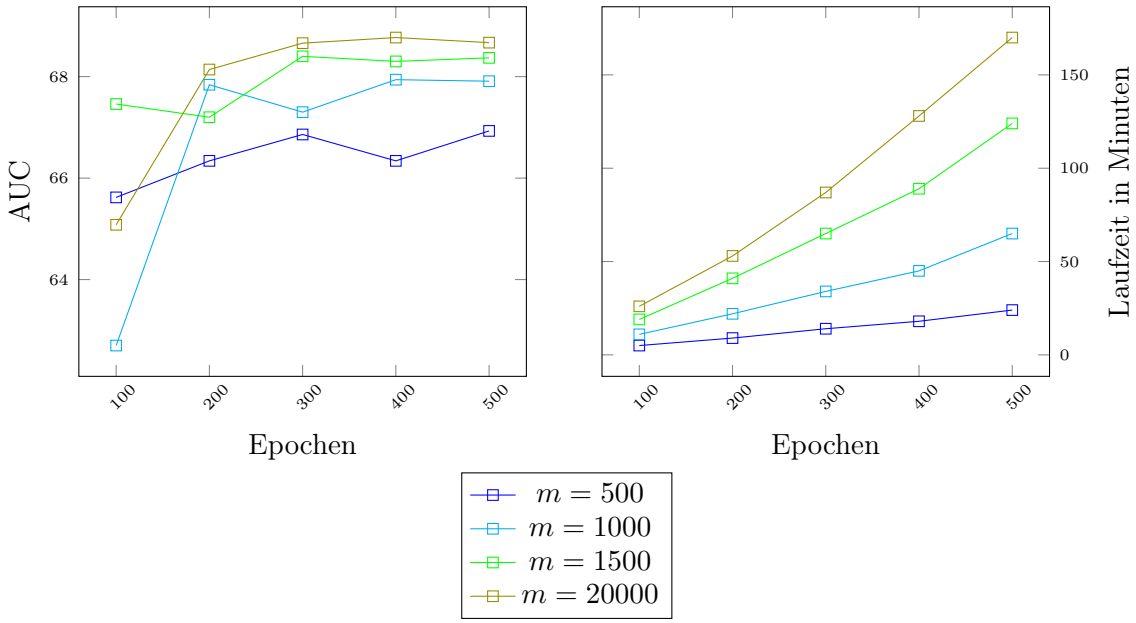


Abbildung 9.2: Konvergenzverhalten - SGD Klassifikator - Arccos Kernel

Fazit

Durch die Anwendung des LLSVM-Algorithmus ist es gelungen auf vielen FACT-Daten kernelbasiert zu lernen. So war es möglich, 170000 FACT-Beispiele mit einem durchschnittlichen *AUC*-Wert von 68.71% unter 8 Stunden zu lernen und zu testen. Dabei wurde deutlich, dass sich der Arccos-Kernel besser für die Klassifikation der Luftschauer eignet. Zum einen ermöglichte der Arccos-Kernel ein höheres Performanz Niveau, zum anderen ließ sich die Arccos-Methode effizienter evaluieren. Störend kam beim RBF-Kernel die zusätzliche Parameteroptimierung für den γ-Parameter hinzu.
Die erfolgreiche Generierung von approximierten linearen FACT Modellen zeigen, dass ressourcenfordernde Aufgaben effizient durch LLSVM gelöst werden können. Dabei ermöglicht die Setzung des Parameters m den Zielkonflikt zwischen Performanz und benötigten Ressourcen erfolgreich zu managen. Somit stellt die Anwendung von LLSVM einen wichtigen Schritt dar, damit lineare Methoden für die Klassifikation von nicht linear-trennbaren Daten gerüstet sind.

Kapitel 10

Funktionsabtastung im Kontext von Kernelmethoden

Die erfolgreiche Anwendung des LLSVM-Algorithmus auf den MNIST und FACT Datensatz verdeutlichen, wie die Nyström Approximation verwendet werden kann, um kernelbasiertes Lernen auf großen Datensätzen zu ermöglichen. Ebenfalls wurde offensichtlich, dass es essentiell für eine erfolgreiche Anwendung ist, den Parameter m auf einen kleinen Wert setzen zu können, ohne dass die Performanz der Klassifikation signifikant verschlechtert wird. Es besteht daher die Anforderung mit einem möglichst kleinen m die Kernel-Matrix hinreichend genau zu approximieren. Weitergehend wird in diesem Kapitel die Genauigkeit von approximierten Kernel-Matrizen untersucht. Wie im Kapitel 6.1 bereits dargelegt, wird die Genauigkeit einer approximierten Matrix von der Auswahl der Spalten/Zeilen sowie dem Rang der zu approximierenden Matrix bestimmt. Insbesondere wird ein Ansatz verfolgt, der auf der Abtastung von Kernelmethoden basiert. Wie man sehen wird, existiert ein Zusammenhang zwischen dem Rang einer Matrix und einer definierten Funktionsabtastung.

10.1 Abtastung von Funktionen im Kontext maschinellen Lernens

Im Allgemeinen versteht man unter einer Funktionsabtastung die Umwandlung von kontinuierlichen Signalen in diskrete Signale. Im Kontext dieser Arbeit wird der Begriff wie folgt verwendet:

Definition. Sei $f : \mathbb{R} \longmapsto \mathbb{R}$ eine kontinuierliche Funktion mit $x \mapsto f(x)$. Weiter sei $\Theta = x_1, \ldots, x_q \in \mathcal{D}(f)$ q Abtastpunkte mit $samples \leftarrow [f(x_1), ..., f(x_q)]$. Dann wird f durch f_q mit $f_q(t) = samples[i] = f(x_i)$ abgetastet, falls für alle $t \in \mathcal{D}(f)$

$$i = \arg\min\{|x_1 - t|, ..., |x_q - t|\} \tag{10.1}$$

gilt.

Unter der Voraussetzung, dass der Index i für den 10.1 gilt, effizient berechnet wurde, ist es möglich durch Substitution von f durch f_q die Laufzeit zu reduzieren. Die Größe der Laufzeitreduzierung hängt davon ab, wie teuer die Funktionsevaluation von f ausfallen würde, die durch f_q reduziert wird.
Die Qualität einer Abtastung, dass heißt die Genauigkeit mit der f durch f_q approximiert wird, wird von den definierten Abtastpunkten bestimmt, an denen f ausgewertet wurde. Die Abtastpunkte sollen so über den Definitionsbereich von f definiert werden, dass eine hinreichend akkurate Abbildung des Funktionsverhaltens von f ermöglicht wird. Es empfiehlt sich Funktionen mit kompakten Definitionsbereichen abzutasten, da für kompakte Intervalle tendenziell weniger Abtastpunkte benötigt werden, um f hinreichend zu approximieren. Ebenfalls wird die Qualität einer Abtastung, durch die gewählte Verteilung der Abtastpunkte, auf dem abzutastenden Intervall beeinflusst. Es gibt eine Vielzahl an Möglichkeiten Abtastpunkte in einem Intervall zu definieren. In dieser Arbeit wird sich darauf beschränkt, ein Intervall $[a, b]$ mit q Abtastpunkten gleichmäßig abzudecken. Das bedeutet, für $[a, b]$ wird $\Theta = (x_1, ..., x_q)$ definiert, wobei $x_1 = a$ und $x_q = b$ ist. Außerdem gilt für alle $i \in \{1, ..., q-1\}$, dass $|x_i - x_{i+1}|$ der gleiche Abstand m existiert.
In Kontext des kernelbasierten Lernens wird die Funktionsabtastung verwendet, um die Kernel-Evaluation zu beschleunigen.

10.1.1 Abtastung Arccos Kernel

Beim Arccos-Kernel bietet sich die Funktionsabtastung von $J_1(arccos(t))$ mit $J_1(\arccos(t)) = \sin\arccos(t) + (\pi - \arccos(t))cos\arccos(t)$ an, wobei $t = \frac{x \cdot y}{\|x\|\|y\|}$ für $k(x, y)$ ist. Die arccos Funktion besitzt einen Definitionsbereich von $[-1, 1]$. Da für jedes $t \in [-1, 1]$ gilt, folgt daraus das abzutastende Intervall von $[-1, 1]$.
Die J_1 Funktion wird durch J_{1_q} abgetastet. Die Funktionswerte lassen sich effizient durch $samples[idx]$ abrufen, wobei idx berechnet wird durch:

$$idx(t) = \begin{cases} \lfloor (1+t) \cdot \lfloor (q/2) \rfloor + 0.5 \rfloor & \text{wenn } q \text{ ungerade ist} \\ \lfloor (1+t) \cdot (q - 1/2) + 0.5 \rfloor & \text{wenn } q \text{ gerade ist} \end{cases} \tag{10.2}$$

Zusammengefasst wird die abgetastete Arccos-Kernelmethode k_q durch

$$k_q(\mathbf{x}, \mathbf{y}) = \frac{1}{\pi}\|\mathbf{x}\|\|\mathbf{y}\| J_{1_q}(t) \tag{10.3}$$

definiert.

10.1.2 Abtastung RBF Kernel

Die Abtastung des RBF-Kernels fällt schwieriger aus. Offensichtlich lässt sich nur sinnvollerweise die Exponentialfunktion abtasten. Wie in 5.5.1 gesehen, liegt der Definitionsbereich in $(-\infty, 0]$. Eine komplette Abtastung des Definitionsbereichs ist daher nicht möglich oder erstrebenswert. Stattdessen braucht es eine Analyse, welche Wertebereiche für t angenommen werden können. Es ist leicht zu sehen, dass der γ-Parameter die Verteilung der Argumente für die Exponentialfunktion stark beeinflusst. Daher lässt sich ein abzutastendes Intervall nur in Abhängigkeit zum γ-Parameter definieren. Hohe Werte für γ resultieren in sehr breite Intervalle. Analog wird das abzutastende Intervall für kleinere γ-Werte schmaler.
In dieser Arbeit wird der RBF-Kernel unter Verwendung von skalierten Daten mit einem festen γ-Wert von $\gamma = 0.01$ abgetastet. Dieses Setting ermöglicht eine kompakte Abtastung durch das Definieren von q gleichmäßig verteilten Abtastpunkten in $[-4, 0]$. Analog zur Abtastung des Arccos-Kernels folgt:

$$k_q = \exp_q(t) \tag{10.4}$$

wobei $\exp_q(t) = samples[idx]$ für

$$idx(t) = \begin{cases} \lfloor (4+t) \cdot \lfloor (q/4) \rfloor + 0.5 \rfloor & \text{wenn } q \text{ ungerade ist} \\ \lfloor (4+t) \cdot (q - 1/4) + 0.5 \rfloor & \text{wenn } q \text{ gerade ist} \end{cases} \tag{10.5}$$

gilt.

Die Abfrage von *samples* durch 10.2 und 10.5 erfüllt 10.1 und ist effizient zu berechnen. Zu beachten ist, dass 10.2 und 10.5 nur für die Intervalle $[-1, 1]$ beziehungsweise $[-4, 0]$ definiert sind. Die Korrektheit der Abfrage wurde über alle in dieser Arbeit betrachteten Abtastungen, mit jeweils 100000 simulierten Werten für t in $[-1, 1]$ beziehungsweise $[-4, 0]$, getestet.

10.2 Evaluation - Abtastungen von Kernelmethoden

Die beschriebene Abtastung des RBF- und Arccos-Kernels kann zu einer Laufzeitreduzierung bei der Erstellung von Kernel-Matrizen führen. Zwar werden zusätzlich $\mathcal{O}(q)$ Speicher für q Abtastpunkte benötigt, jedoch ist dies vernachlässigbar, solange q keine sehr großen Werte annimmt.
In dieser Arbeit wird die Abtastung einer Kernelmethode k durch k_q nicht primär aus der Perspektive der Laufzeitreduzierung betrachtet werden. Stattdessen wird der Fokus auf die

Eigenschaften der resultierenden Matrix K_q liegen. Hierbei wird K_q analog zu 5.2 durch $K_q := (k_q(x_i, x_j))_{i,j}$ definiert.

Grundlegende Versuchsanordnung

Anders wie in den vorangegangenen Experimenten, werden im Folgenden Problemstellungen der Form 5.7 mit quadratischen Methoden gelöst. Dafür wird die populäre Klassifikationsmethode *LIBSVM* verwendet, welche eine Anwendung von vorberechneten Kernel-Matrizen erlaubt [10]. Auf das Lernen von sehr großen Datensätzen mit Einsatz von Linearisierungen wird zunächst verzichtet. Stattdessen wird *LIBSVM* auf jeweils einer Trainings- und Testmenge mit 10000 Beispielen angewendet.

10.2.1 Eigenschaften abgetasteter Kernel-Matrizen

In dieser Versuchsreihe werden die resultierenden Matrizen K_q anhand ihrer Eigenwerte untersucht. Dabei werden die Eigenwerte Λ durch $K_q = U\Lambda U^T$ ermittelt und anhand ihres Vorzeichens analysiert. Ein Eigenwert $\lambda_i \in \Lambda$ wird für ein kleines ϵ in folgende Klassen eingeteilt:

- positiv: wenn $\lambda_i > \epsilon$
- negativ: wenn $\lambda_i < \epsilon$
- null: wenn $|\lambda_i| <= \epsilon$

Die Anzahl der Eigenwerte werden gemäß ihrer Klassen in *pos*, *neg* und *zero* gespeichert.

Die Versuchsergebnisse für den MNIST 10.1, 10.2 und FACT Datensatz 10.3, 10.4 zeigen einheitlich, dass es einen Zusammenhang zwischen der Anzahl von positiven Eigenwerten und der gewählten Abtastung durch q gibt. Offensichtlich wird beim Verwenden von Abtastungen, ein Teil der positiven Eigenwerte von K_q negativ. Der Effekt wird durch die Wahl von gröberen Abtastungen (wenige Abtastpunkte) verstärkt. Betrachtet man die Verteilung der Eigenwerte genauer, so sind Unterschiede zwischen den betrachteten Fällen erkennbar. Während der Rang bei den Versuchen auf dem MNIST Datensatz für alle definierten Abtastungen gleich bleibt, kommt es beim FACT-Datensatz unter Verwendung des RBF-Kernels zu einer Rangreduzierung in Abhängigkeit von q.
Durch das Entstehen von negativen Eigenwerten in abgetasteten Kernel-Matrizen, ist K_q nicht positiv semidefinit. Folglich handelt es sich bei K_q um keine Kernel-Matrix gemäß 5.2.
Bei der Analyse der Laufzeit für K_q auf den MNIST Daten 10.1 und den FACT Daten 10.2, zeigt sich, dass durch die Abtastung des RBF-Kernels keine Laufzeitreduzierung erreicht werden konnte. Anders fällt die Analyse der Laufzeit für den Arccos-Kernel aus. Dort

MNIST Datensatz

	(pos,zero,neg)	Rang
K	(10000,0,0)	10000
K_{20}	(5362,0,4638)	10000
K_{50}	(5739,0,4261)	10000
K_{100}	(6188,0,3812)	10000
K_{150}	(6535,0,3465)	10000
K_{200}	(6825,0,3175)	10000
K_{250}	(7072,0,2928)	10000
K_{300}	(7287,0,2713)	10000
K_{400}	(7642,0,2358)	10000
K_{600}	(8161,0,1839)	10000
K_{20000}	(9874,0,26)	10000

Tabelle 10.1: RBF-Kernel

	(pos,zero,neg)	Rang
K	(10000,0,0)	10000
K_{20}	(5182,0,4818)	10000
K_{50}	(5340,0,4660)	10000
K_{100}	(5538,1,4462)	10000
K_{150}	(5700,0,4300)	10000
K_{200}	(5841,0,4159)	10000
K_{250}	(5972,0,4028)	10000
K_{300}	(6088,0,3912)	10000
K_{400}	(6303,0,3697)	10000
K_{600}	(6666,0,3334)	10000
K_{20000}	(10000,0,0)	10000

Tabelle 10.2: Arccos-Kernel

FACT Datensatz

	(pos,zero,neg)	Rang
K	(9478,522,0)	9138
K_{20}	(159,9703,138)	297
K_{50}	(337,9361,302)	639
K_{100}	(641,8769,590)	1231
K_{150}	(918,8215,867)	1785
K_{200}	(1197,7668,1135)	2332
K_{250}	(1446,7186,1368)	2813
K_{300}	(1691,6700,1609)	3300
K_{400}	(2159,5791,2050)	4208
K_{600}	(2988,4154,2858)	5845
K_{20000}	(5412,74,4514)	9926

Tabelle 10.3: RBF-Kernel

	(pos,zero,neg)	Rang
K	(9925,75,0)	9925
K_{20}	(6110,75,3815)	9925
K_{50}	(6701,75,3224)	9925
K_{100}	(7466,75,2459)	9925
K_{150}	(8061,75,1864)	9925
K_{200}	(8537,75,1388)	9925
K_{250}	(8917,75,1008)	9925
K_{300}	(9217,75,708)	9925
K_{400}	(9629,75,296)	9925
K_{600}	(9914,75,11)	9925
K_{20000}	(9925,75,0)	9925

Tabelle 10.4: Arccos-Kernel

konnte mit K_q eine deutliche Reduzierung der Laufzeit gegenüber K erzielt werden. Dies lässt sich durch das Abtasten des J_1 Ausdrucks erklären, der mehrere Funktionsaufrufe umfasst, während bei der Abtastung des RBF-Kernels nur die Exponentialfunktion abgetastet wurde. Für die FACT-Daten wird weniger als die Hälfte der Zeit für die Berechnung des abgetasteten Arccos-Kernels benötigt. Zudem zeigen die Ergebnisse, dass mit steigendem q die Laufzeit konstant bleibt. Die Ursache liegt in der verwendeten Implementierung,

die in A diskutiert wird. Soll der Arccos-Kernel durch Abtastungen beschleunigt werden, so bietet sich folglich eine Abtastung mit vielen Abtastpunkten an. Auf diesem Weg wird von der Laufzeitreduzierung profitiert, ohne störende Begleiterscheinungen, in Form von negativen Eigenwerten, hervorzurufen.

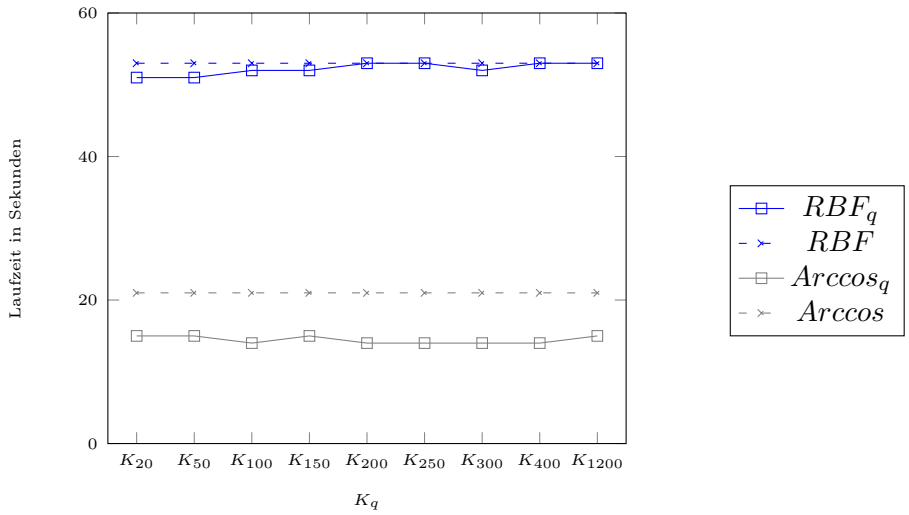


Abbildung 10.1: Laufzeit MNIST

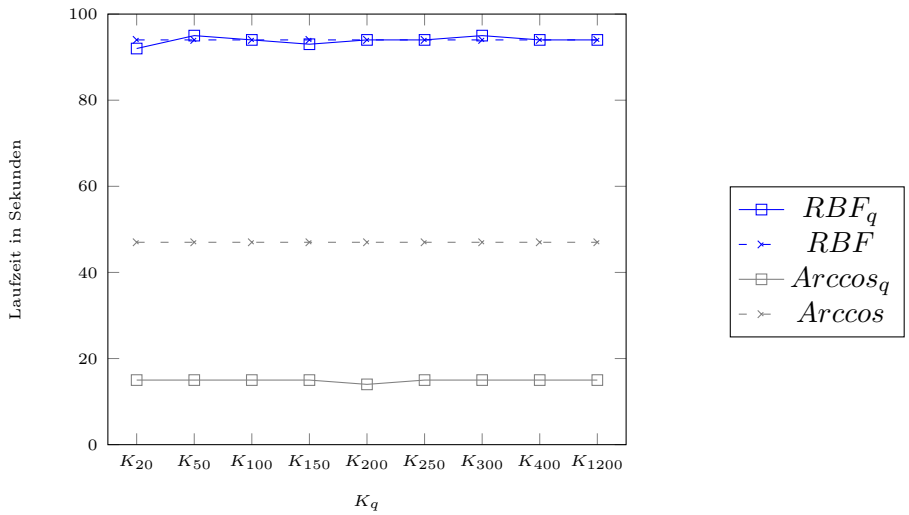


Abbildung 10.2: Laufzeit FACT

Generierung von rang-reduzierten Matrizen

Die Experimente zeigten, dass eine Abtastung mit q Abtastpunkten die Anzahl der positiven Eigenwerte in K_q beeinflussen. Insbesondere entstanden abgetastete Matrizen mit negativen Eigenwerten. Die durch die Abtastung induzierten negativen Eigenwerte können problematisch werden. Zum einen könnten sie sich negativ auf die Performanz auswirken,

zum anderen stellen die negativen Eigenwerte ein Hindernis bei der Nyström Approximation dar, da diese nur für psd-Matrizen definiert ist. Ein einfacher Ansatz, um eine psd-Matrix zu generieren, ist die Anwendung einer Eigenwertkorrektur, die negative Eigenwerte auf null setzt. Zusätzlich findet durch das null setzten der negativen Eigenwerte eine Rangreduzierung der Eigenwert-korrigierten Matrix statt.

10.2.1 Definition. Sei $K_q = U\Lambda U^T$ eine abgetastete Kernel Matrix. Dann ist $K_{LOWRANK_q}$ eine Eigenwert-korrigierte Matrix von K_q definiert durch $K_{LOWRANK_q} = U corrected(\Lambda) U^T$ wobei *corrected* die negativen Eigenwerte null setzt.

Die $K_{LOWRANK_q}$ Matrizen weisen eine Rangreduzierung in Abhängigkeit von q auf. Tendenziell fällt die Rangreduzierung für Abtastungen mit kleinen q größer aus. Die Perfor-

Rang	*MNIST*		*FACT*	
	RBF	*Arccos*	*RBF*	*Arccos*
$K_{LOWRANK_{20}}$	5362	5182	159	6110
$K_{LOWRANK_{50}}$	5739	5340	639	6701
$K_{LOWRANK_{100}}$	6188	5538	641	7466
$K_{LOWRANK_{150}}$	6535	5700	918	8061
$K_{LOWRANK_{200}}$	6825	5841	1197	8537
$K_{LOWRANK_{300}}$	7287	6088	1691	9217
$K_{LOWRANK_{400}}$	7642	6303	2159	9629
$K_{LOWRANK_{600}}$	8161	6666	2988	9914
$K_{LOWRANK_{400}}$	7642	6303	2159	9629
$K_{LOWRANK_{20000}}$	9974	10 000	5486	9925

Tabelle 10.5: Rang von Lowrank Matrizen

manz der generierten $K_{LOWRANK_q}$ Matrizen ist im Vergleich zu K_q für grob gewählte Abtastungen ($q <= 100$) besser. Dies liegt vermutlich an den Umstand, dass für kleine q die Matrix K_q mehr negative Eigenwerte aufweist, die die Performanz negativ beeinflussen. Mit steigenden Abtastpunkten konvergiert die Performanz von K_q und $K_{LOWRANK_q}$ zum Performanz-Niveau von K.

10.2.2 Approximieren von Lowrank Matrizen

Nach dem nun dargelegt wurde, dass eine Kernel-Abtastung mit anschließender Eigenwert-Korrektur genutzt werden kann, um rang-reduzierte Matrizen zu induzieren, wird nun analysiert, mit welcher Genauigkeit $K_{LOWRANK_q}$ durch Nyström approximiert werden kann. Insbesondere wird evaluiert, ob die Rangreduzierung eine genauere Approximation

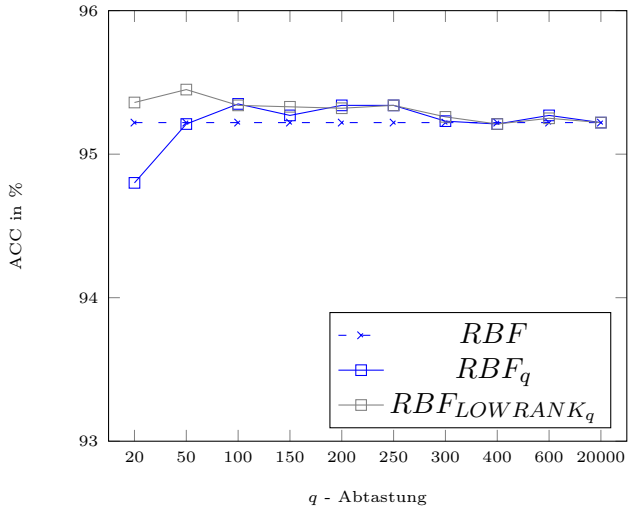


Abbildung 10.3: Performanz: RBF - RBF_q - $RBF_{LOWRANKq}$

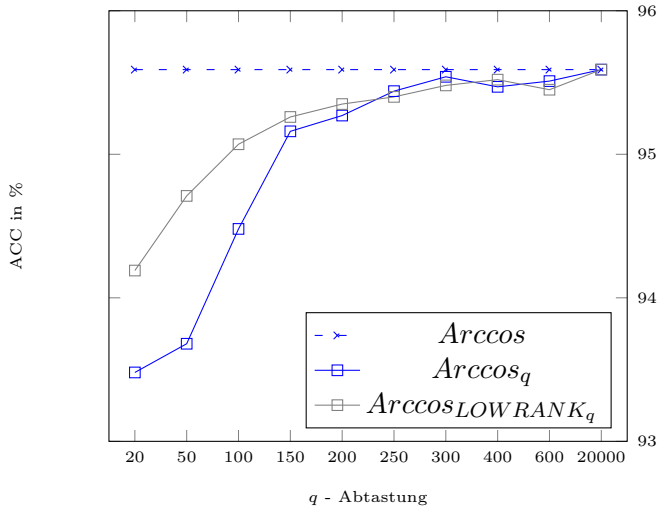


Abbildung 10.4: Performanz: $Arccos$ - $Arccos_q$ - $Arccos_{LOWRANKq}$

ermöglicht. Die Nyström Approximation einer Matrix $K^{n \times n}$ wird unter zwei Selection Strategien für m ausgeführt:

- Random-Selection: Die Ziehung von m gleich-verteilten Zufallszahlen aus $[1, ..., n]$. Dies stellt die bereits bekannte und verwendete Methode dar.
- Best-Selection: Es werden die Indizes der m größten Eigenwerte von K gezogen.

Im Folgenden wird die Anwendung der Nyström Approximation gemäß 6.1 unter Anwendung der beiden Auswahlstrategien angewendet.
Als Notation wird für Parameter m

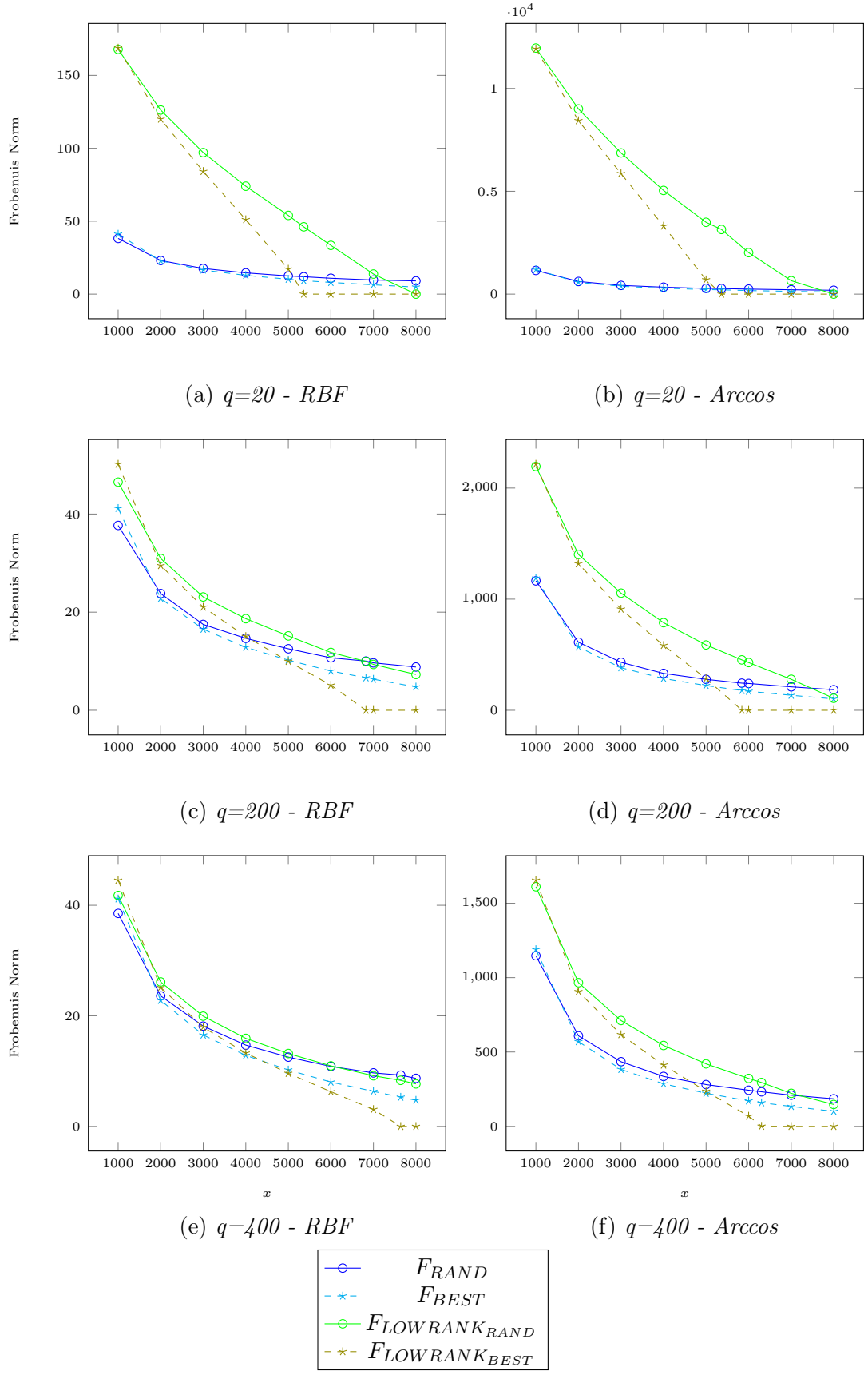
- $Nyst(K, m)_{RAND}$ - Nyström Approximation von K unter Anwendung der Random-Selection

- $Nyst(K,m)_{BEST}$ - Nyström Approximation von K unter Anwendung der BEST Selection

eingeführt.
Die Genauigkeit einer Approximation $\tilde{K}$ wird durch die Abweichung zu K ermittelt. Es werden folgende Fälle untersucht:

- $F_{RAND} = \|K - Nyst(K,m)_{RAND}\|_F$
- $F_{BEST} = \|K - Nyst(K,m)_{BEST}\|_F$
- $F_{LOWRANK_{RAND_q}} = \|K_{LOWRANK_q} - Nyst(K_{LOWRANK_q},m)_{RAND}\|_F$
- $F_{LOWRANK_{BEST_q}} = \|K_{LOWRANK_q} - Nyst(K_{LOWRANK_q},m)_{BEST}\|_F$.

Die Ergebnisse wurden für den MNIST Datensatz erhoben. Die unterschiedlichen Fälle wurden für den Arccos- und RBF-Kernel ermittelt, wobei die Lowrank-Matrizen für $q = \{20, 200, 400\}$ definiert wurden. In 10.5 sind die Abweichungen der jeweiligen Approximation in Abhängigkeit zu m gegeben.
Nicht überraschend zeigen die Ergebnisse, dass mit steigendem Wert für m die Genauigkeit der approximierten Matrizen steigt. Für den Fall, dass die Random-Selection gewählt wurde, approximierten die $LOWRANK_q$ Matrizen deutlich schlechter, wie $Nyst(K,m)_{RAND}$. Erst mit aufsteigendem m näherten sich F_{RAND} und $F_{LOWRANK_{RAND_q}}$ hinsichtlich ihrer Genauigkeit an, wobei für $m = 8000$ die $K_{LOWRANK_q}$ Matrizen geringfügig genauer approximiert wurden.
Wurden die Spalten/Zeilen nach der Best-Selection Strategie gewählt, dann profitiert $K_{LOWRANK_q}$ durch eine exakte Approximation wenn $m >= Rang(K_{LOWRANKq})$ ist. Falls für $m << Rang(K_{LOWRANKq})$ gilt, stellt $Nyst(K,m)_{RAND}$ und $Nyst(K,m)_{BEST}$ exaktere Approximationen dar.
In jedem betrachteten Fall ab $m > 1000$ liefert die Nyström Approximation unter der BES-Selection eine exaktere Approximation. Allerdings sind die Unterschiede bei F_{RAND} und F_{BEST} nur geringfügig. Der Einfluss der Selection-Strategie auf die Genauigkeit der Approximation, ist bei rang-reduzierten Matrizen ungleich höher. Dies lässt sich dadurch erklären, dass eine rang-reduzierte Matrix $K_{LOWRANK_q}$ eine größere Anzahl an Spalten/Zeilen aufweist, die linear abhängig sind. Folglich steigt die Wahrscheinlichkeit bei der Random-Selection linear abhängige Spalten/Zeilen zu wählen, was sich negativ auf die Genauigkeit auswirkt. Die Ergebnisse verdeutlichen, dass der Rang einer zu approximierender Matrix nicht als alleiniger Einflussfaktor auf die Genauigkeit der Approximation gesehen werden kann. Stattdessen steigt, mit fallendem Rang, die Bedeutung der Landmark-Selection.



(a) *q=20 - RBF* (b) *q=20 - Arccos*

(c) *q=200 - RBF* (d) *q=200 - Arccos*

(e) *q=400 - RBF* (f) *q=400 - Arccos*

Abbildung 10.5: MNIST - Genauigkeit Approximation K - $K_{LOWRANK}$

10.3 Eigenwert korrigierte Nyström Approximation

Damit eine rang-reduzierte Matrix gemäß 10.2.1 generiert werden kann, muss eine Eigenwertkorrektur auf der kompletten Matrix durchgeführt werden. Die dafür benötigte Eigendekomposition für eine $n \times n$ Matrix liegt in $\mathcal{O}(n^3)$. Folglich ist der Einsatz von Lowrank Matrizen auf großen Datensätzen nicht praktikabel. Ein Lösungsansatz ist die Eigenwertkorrektur nur für einen quadratischen Teilbereich der Matrix durchzuführen. Bezogen auf die Nyström Approximation bietet sich eine Eigenwertkorrektur von K_{mm} an. Entsprechend wird die Eigenwert-korrigierte Nyström Approximation $\tilde{K}$ für q durch

$$\tilde{K}_{Lowrank_q} = K_{nm_q} K^{+}_{mm_{LOWRANK_q}} K_{mn_q} \tag{10.6}$$

definiert, wobei $^+$ das Pseudoinverse über die generierte Lowrank-Matrix von K_{mm} darstellt. Durch begrenzen der Eigenwertkorrektur auf K_{mm}, lässt sich der Ansatz effizient in der Anwendung der Nyström Approximation implementieren.
In 10.6 wurde die erreichte Performanz für $m = 1000$ unter Verwendung von $\tilde{K}_{Lowrank_q}$ in Abhängigkeit zu q untersucht. Das Performanz-Niveau für $\tilde{K}_{Lowrank_q}$ ist mit der klassischen Nyström Approximation vergleichbar. Die erreichte Performanz des RBF-Kernels liegt im Durchschnitt minimal unterhalb der Performanz von $\tilde{K}$. Wiederum ist für $100 <= q < 20000$ die erreichte Performanz beim Verwenden des Arccos Kernels und von $\tilde{K}_{Lowrank_q}$ ein wenig besser, als bei der nicht modifizierten Approximation. Auf Grundlage der Ergebnisse in 10.5 ist nicht davon auszugehen, dass die leichte Verbesserung der Performanz, auf eine effizientere Approximation durch induzierter Rangreduzierung zurückzuführen ist.

	RBF		Arccos	
	ACC	sd	ACC	sd
$K_{LOWRANK_{20}}$	95.4	0.1248	93.62	0.1121
$K_{LOWRANK_{50}}$	95.55	0.1511	94.42	0.1735
$K_{LOWRANK_{100}}$	95.65	0.083	94.81	0.1056
$K_{LOWRANK_{150}}$	95.65	0.0971	94.91	0.1774
$K_{LOWRANK_{200}}$	95.63	0.0909	94.94	0.2239
$K_{LOWRANK_{250}}$	95.66	0.0787	95.04	0.2148
$K_{LOWRANK_{300}}$	95.68	0.0743	94.96	0.192
$K_{LOWRANK_{400}}$	95.67	0.0731	95.05	0.2237
$K_{LOWRANK_{600}}$	95.72	0.1424	94.86	0.1465
$K_{LOWRANK_{20000}}$	95.69	0.1046	94.64	0.2118
K	95.81	0.4747	94.64	0.2169

Tabelle 10.6: Performanz $Nyström_{LOWRANK_q}$ auf dem $MNIST$ Datensatz für $m = 1000$

Zusammenfassung

Die Versuchsreihe 10.5 deutet darauf hin, dass durch eine Funktionsabtastung induzierte rang-reduzierte Matrix, nicht effizienter durch Nyström approximiert wird. Zwar hielten die generierten Lowrank-Matrizen ein zufriedenstellendes Performanz-Niveau 10.3,10.4, allerdings wiesen sie ein schlechteres Approximationsverhalten auf. Einzig unter den Voraussetzungen, dass der Parameter $m >= Rang$ gesetzt wurde und die Best-Selection Strategie gewählt wurde, konnten die Matrizen von der Rangreduzierung profitieren. Beide Voraussetzungen stehen im Widerspruch zu den Anforderungen von Matrizen Approximationen. So erfordern die induzierten Lowrank-Matrizen eine leistungsfähige Landmark-Selection Strategie, die allerdings für viele Daten skalierbar bleiben muss. Die Setzung von $m >= Rang$ ist ebenfalls nicht praktikabel, da bereits häufig ein ausreichendes Performanz-Niveau für $m << Rang$ erreicht wird. Zusammengefasst scheinen die notwendigen Kosten, um von einer Lowrank-Matrix profitieren zu können, in keinem Verhältnis zur exakteren Approximation zu stehen. Ein Grund dafür, liegt in der 'Gutmütigkeit' von Kernelmethoden hinsichtlich von Manipulationen. So zeigte sich in vielen Fällen, dass die Abtastung von Kernelmethoden zu keiner drastischen Verschlechterung der Performanz führte. Bezüglich der Approximation, wirkt sich eine leicht akkuratere Approximation der Kernel Matrix, in der Regel nicht signifikant auf die erreichte Performanz aus.
Dies lässt sich auch als Erklärungsmuster für die erfolgreiche Anwendung des LLSVM-Algorithmus auf den FACT-Daten sehen. So war es möglich den Parameter m auf ein hinreichend kleinen Wert zu setzen, so dass die Versuche effizient durchgeführt werden konnten und ein zufriedenstellendes Performanz-Niveau erreicht wurde. Über die Setzung von m war es möglich, den Zielkonflikt zwischen der Genauigkeit der approximierten Matrix und dem benötigten Aufwand zu steuern. Dadurch konnten linearisierte Modelle, in Abhängigkeit zu den gegebenen Ressourcen und Zielsetzungen, flexibel durch LLSVM generiert werden.

Ausblick

Im Hinblick auf die Ergebnisse, scheint eine effizientere Nyström Approximation durch Funktionsabtastungen nicht praktikabel zu sein. Eventuell könnte die gezielte Funktionsabtastung und Korrektur der Eigenwerte verwendet werden, um Matrizen durch den gefallenen Rang (via Eigendekomposition) kompakter zu repräsentieren. Folglich würde der Ansatz eher einer Komprimierung von Matrizen entsprechen. Im Kontext vom maschinellen Lernen wäre der Anwendungsbereich sehr begrenzt.

Ausgehend von den Unterschieden in der erreichten Performanz beim RBF- und Arccos-Kernel, ist die Wahl der Kernelmethode ein naheliegender Ansatz zur Verbesserung der

FACT-Luftschauer Klassifikation. Die Familie der Arccos-Kernelmethode bietet das Potential Multi-Layer Netze zu simulieren [11]. Unter der Voraussetzung, dass die Multi-Layer Strukturen effizient berechnet werden können, könnte die Klassifikation der FACT-Versuche verbessert werden.
Ebenfalls könnten die LLSVM-Modelle auf echten Luftschauern getestet werden.

Anhang A

Implementierung

Für die durchgeführten Experimente wurden Methoden aus dem Python Framework *scikit-learn* verwendet [23]. Das Framework basiert auf *NumPy* beziehungsweise *SciPy* Methoden, und bietet eine Vielzahl an Funktionen im Kontext des maschinellen Lernens [7].
Für die durchgeführten Versuche wurde der *scikit-learn* SGD-Klassifikator im Single-Core Betrieb angewendet. Die unterschiedlichen Skalierungen wurden ebenfalls durch *scikit-learn* realisiert.
Der LLSVM-Algorithmus konnte effizient durch den Einsatz von Parallelisierung ausgeführt werden. Die Berechnung von 6.4 und 6.5 können über weite Bereiche unabhängig berechnet werden und eignen sich somit für eine parallele Ausführung. In der Arbeit wurde die Parallelität durch Threads realisiert.
In 10.2.1 wurde festgestellt, dass die Laufzeit von Abtastungen unabhängig von q konstant bleibt. Dies lässt sich durch das effiziente Zugriffsverhalten von *NumPy* Arrays begründen. Des Weiteren wurde bei der Evaluation deutlich, dass der RBF-Kernel, im Vergleich zum Arccos-Kernel, weniger effizient evaluiert werden konnte. Als Grund für die Effizienzunterschiede wurde das Berechnen der quadratischen euklidischen Distanz für große Eingaben identifiziert. Hierfür wurde die entsprechenden Funktion aus *NumPy* und *SciPy* getestet, welche aber das Effizienzproblem für große Eingaben nicht lösen konnten.

Literaturverzeichnis

[1] *Photon-Stream*. `https://github.com/fact-project/photon_stream`, 2018.

[2] Anderhub, H, M Backes, A Biland, V Boccone, I Braun, T Bretz, J Buss, F Cadoux, V Commichau, L Djambazov, D Dorner, S Einecke, D Eisenacher, A Gendotti, O Grimm, H von Gunten, C Haller, D Hildebrand, U Horisberger, B Huber, K S Kim, M L Knoetig, J H Köhne, T Krähenbühl, B Krumm, M Lee, E Lorenz, W Lustermann, E Lyard, K Mannheim, M Meharga, K Meier, T Montaruli, D Neise, F Nessi-Tedaldi, A K Overkemping, A Paravac, F Pauss, D Renker, W Rhode, M Ribordy, U Röser, J P Stucki, J Schneider, T Steinbring, J Thaele F Temme, S Tobler, G Viertel, P Vogler, R Walter, K Warda, Q Weitzel und M Zänglein: *Design and operation of FACT - the first G-APD Cherenkov telescope*. Journal of Instrumentation, 8(06):P06008, 2013.

[3] Aronszajn, N.: *Theory of Reproducing Kernels*. Transactions of the American Mathematical Society, 1950.

[4] Bottou, Léon: *Large-Scale Machine Learning with Stochastic Gradient Descent*. In: Lechevallier, Yves und Gilbert Saporta (Herausgeber): *Proceedings of COMPSTAT'2010*, Seiten 177–186, Heidelberg, 2010. Physica-Verlag HD.

[5] Brügge, Kai, Alexey Egorov, Christian Bockermann, Katharina Morik und Wolfgang Rhode: *Distributed Real-Time Data Stream Analysis for CTA*. In: *Astronomical Data Analysis Software and Systems (ADASS XXVII)*, 2017. Accepted for publication.

[6] Brügge, Kai, Alexey Egorov, Christian Bockermann, Katharina Morik und Wolfgang Rhode: *Distributed Real-Time Data Stream Analysis for CTA*. In: *Astronomical Data Analysis Software and Systems (ADASS XXVII)*, 2017. Accepted for publication.

[7] Buitinck, Lars, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer,

Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt und Gaël Varoquaux: *API design for machine learning software: experiences from the scikit-learn project.* In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, Seiten 108–122, 2013.

[8] Buss, Jens, M.L. Ahnen, M. Balbo, M. Bergmann, Adrian Biland, Christian Bockermann, Thomas Bretz, K. A. Brügge, Daniela Dorner, Sabrina Einecke, Jan Freiwald, Christina Hempfling, D. Hildebrand, Gareth Hughes, Werner Lustermann, Karl Mannheim, K. Meier, Katharina Morik, Sebastian Müller, Dominik Neise, A. Neronov, Max Nöthe, Wolfgang Rhode, Fabian Temme, Julia Thaele, Simona Toscano, Patrick Vogler, Roland Walter und A. Wilbert: *FACT – Influence of SiPM Crosstalk on the Performance of an Operating Cherenkov Telescope.* In: *Proceedings, 34th International Cosmic Ray Conference (ICRC 2015)*, Band 34 der Reihe *International Cosmic Ray Conference*, Seite 863, The Hague, July 2015.

[9] Campbell, C.: *Radial Basis Function Networks 1.* Kapitel An Introduction to Kernel Methods, Seiten 155–192. Physica Verlag Rudolf Liebing KG, 2001.

[10] Chang, Chih-Chung und Chih-Jen Lin: *LIBSVM: A Library for Support Vector Machines.* ACM Trans. Intell. Syst. Technol., 2(3):27:1–27:27, 2011.

[11] Cho, Youngmin und Lawrence K. Saul: *Kernel Methods for Deep Learning.* In: Bengio, Y., D. Schuurmans, J. D. Lafferty, C. K. I. Williams und A. Culotta (Herausgeber): *Advances in Neural Information Processing Systems 22*, Seiten 342–350. Curran Associates, Inc., 2009.

[12] Drineas, Petros und Michael W. Mahoney: *On the Nyström Method for Approximating a Gram Matrix for Improved Kernel-Based Learning.* The Journal of Machine Learning Research, 6:2153–2175, Dezember 2005.

[13] Fine, Shai und Katya Scheinberg: *Efficient Svm Training Using Low-rank Kernel Representations.* The Journal of Machine Learning Research, 2:243–264, März 2002.

[14] Hofmann, Thomas und Bernhard Schölkopf: *Kernel Methods in Machine Learning.* The Annals of Statistics, 2008.

[15] Hsu, Chih-Wei, Chih-Chung Chang und Chih-Jen Lin: *A Practical Guide to Support Vector Classification.* Technischer Bericht, Department of Computer Science, National Taiwan University, 2003.

[16] Joachims, Thorsten: *Text Categorization with Support Vector Machines: Learning with Many Relevant Features.* In: *Proceedings of the 10th European Conference on Machine Learning*, ECML'98, Seiten 137–142, Berlin, Heidelberg, 1998. Springer-Verlag.

[17] Kumar, Sanjiv, Mehryar Mohri und Ameet Talwalkar: *Sampling Methods for the Nyström Method.* The Journal of Machine Learning Research, 13(1):981–1006, 2012.

[18] Lecun, Yann, Léon Bottou, Yoshua Bengio und Patrick Haffner: *Gradient-based learning applied to document recognition.* In: *Proceedings of the IEEE*, Seiten 2278–2324, 1998.

[19] Liu, Cheng-Lin, Kazuki Nakashima, Hiroshi Sako und Hiromichi Fujisawa: *Handwritten Digit Recognition Using State-of-the-Art Techniques.* In: *Proceedings of the Eighth International Workshop on Frontiers in Handwriting Recognition (IWFHR'02)*, IWFHR '02, Seiten 320–, Washington, DC, USA, 2002. IEEE Computer Society.

[20] Muller, K. R., S. Mika, G. Ratsch, K. Tsuda und B. Scholkopf: *An Introduction to Kernel-based Learning Algorithms.* Transactions on Neural Networks, 12(2):181–201, März 2001.

[21] Neal, Radford M.: *Bayesian Learning for Neural Networks.* Springer-Verlag, Berlin, Heidelberg, 1996.

[22] Osuna, Edgar, Robert Freund und Federico Girosi: *Training Support Vector Machines: an Application to Face Detection.* Seiten 130–136, 1997.

[23] Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot und E. Duchesnay: *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research, 12:2825–2830, 2011.

[24] Srebro, Nathan und Shai Ben-David: *Learning Bounds for Support Vector Machines with Learned Kernels.* In: Lugosi, Gábor und Hans Ulrich Simon (Herausgeber): *Learning Theory*, Seiten 169–183, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[25] Vapnik, V.: *Principles of Risk Minimization for Learning Theory.* In: *Proceedings of the 4th International Conference on Neural Information Processing Systems*, NIPS'91, Seiten 831–838. Morgan Kaufmann Publishers Inc., 1991.

[26] Vapnik, V. und O. Chapelle: *Bounds on Error Expectation for Support Vector Machines.* Neural Computation, 12(9):2013–2036, 2000.

[27] Vapnik, V. N.: *An overview of statistical learning theory.* IEEE Transactions on Neural Networks, 10(5):988–999, Sept 1999.

[28] Vapnik, Vladimir N.: *The Nature of Statistical Learning Theory.* Springer-Verlag, 1995.

[29] Williams, Christopher K. I. und Matthias Seeger: *Using the Nyström Method to Speed Up Kernel Machines.* In: Leen, T. K., T. G. Dietterich und V. Tresp (Herausgeber): *Advances in Neural Information Processing Systems 13*, Seiten 682–688. MIT Press, 2001.

[30] Zhang, Kai, Liang Lan, Zhuang Wang und Fabian Moerchen: *Scaling up Kernel SVM on Limited Resources: A Low-rank Linearization Approach.* In: Lawrence, Neil D. und Mark Girolami (Herausgeber): *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, Band 22 der Reihe *Proceedings of Machine Learning Research*, Seiten 1425–1434. PMLR, 21–23 Apr 2012.

[31] Zhang, Tong: *Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms.* In: *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, Seiten 116–. ACM, 2004.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 28. September 2018

Martin Senz