Fakultät Elektrotechnik, Medien und Informatik

Konzeption und Implementierung eines Frameworks zur Sicherung der Vertraulichkeit bei Neuronalen Netzen

Masterarbeit im Studiengang Künstliche Intelligenz

vorgelegt von

Sabau Patrick

Matrikelnummer 12913724

Erstgutachter:	Prof. Dr.-Ing. Christoph P. Neumann
Zweitgutachter:	Prof. Dr. rer. nat. Daniel Loebenberger

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Selbstständigkeitserklärung

Name und Vorname
der Studentin/des Studenten: **Sabau Patrick**

Studiengang: **Künstliche Intelligenz**

Ich bestätige, dass ich die Masterarbeit mit dem Titel:

Konzeption und Implementierung eines Frameworks zur Sicherung der Vertraulichkeit bei Neuronalen Netzen

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum: 13. Mai 2023

Unterschrift:

Kurzdarstellung

Hier steht eine deutsche Zusammenfassung der Arbeit

Abstract

This is the location of the abstract

Inhaltsverzeichnis

Kapitel 1

Einleitung

Machine Learning ist spätestens seit der Veröffentlichung von ChatGPT[1] im Mainstream angelangt. Dabei handelt es sich um einen Chatbot des Unternehmens OpenAI, welcher mittels natürlicher Sprache mit Nutzern kommuniziert und eine Vielzahl an Aufgaben bewältigen kann. Bereits nach zwei Monaten nutzen über 100 Millionen verschiedene Personen den Chatbot und machen diesen damit zu der am schnellsten wachsenden Plattform überhaupt. Im Hintergrund von ChatGPT läuft ein sogenanntes Large Language Model, welches anhand von menschlichem Feedback optimiert wurde [1].

Jedoch ist Machine Learning bereits seit Jahren in vielen Produkten verankert. Die Venture Capital Gesellschaft FirstMark gibt ein Überblick über Unternehmen, die im Machine Learning Umfeld tätig sind und Produkte, die diese Technologien unterstützen [2].

Eine Voraussetzung für diese Anwendungen sind Daten, viele Daten. Teilweise sind diese Daten privat. Diese werden von Unternehmen gesammelt und dazu genutzt, Produkte zu verbessern oder sogar neue Services zu entwickeln und anzubieten. Beispielsweise nutzen Soziale Medien die Nutzerdaten, um die Reihenfolge von Beiträgen zu sortieren[2]. Jedoch sind diese Daten nicht immer sicher. So wurde beispielsweise über 50 Millionen Profile des Sozialen Netzwerks Facebook ausgelesen, und anschließend wurden diese Personen in Bezug auf die US Wahl 2016 manipuliert [3].

Das Bundesamt für Sicherheit in der Informationstechnik, kurz BSI, definiert das Schutzziel der Vertraulichkeit wie folgt [4]: *"Vertraulichkeit ist der Schutz vor unbefugter Preisgabe von Informationen. Vertrauliche Daten und Informationen dürfen ausschließlich Befugten in der zulässigen Wiese zugänglich sein"*. Da Machine Learning Anwendungen, darunter auch Neuronale Netze, vertrauliche Daten zum Training als auch für eine Vorhersage nutzen, gilt es auch hier, das Schutzziel der Vertraulichkeit zu gewähren. Datenlecks, wie das Beispiel von Facebook [3], oder auch Sammelklagen gegen Machine Learning Modelle, wie das Beispiel GitHub Copilot [5], zeigen, dass hier allerdings noch Nachholbedarf besteht.

[1] https://openai.com/blog/chatgpt
[2] https://github.com/twitter/the-algorithm

Die folgende Arbeit geht auf die Frage ein, wie der Schutz der Vertraulichkeit mit der Nutzung von Daten in Neuronalen Netzen in Einklang gebracht werden kann. Dazu werden zuerst Angriffe beleuchtet, welche die Vertraulichkeit von Neuronalen Netzen gefährden. Anschließend wird eine Reihe von Maßnahmen aufgeführt, die das Ziel haben, die vorher genannten Angriffe unwirksam zu machen. Eine Menge dieser Maßnahmen wird anschließend in einem Framework namens PrivacyFlow gebündelt. Die Konzeption und Implementierung dieses Frameworks wird detailliert geschildert. Nach Zusammenfassen der Ergebnisse, kann eine Handlungsempfehlung erstellt werden. Zum Abschluss der Arbeit werden offene Probleme der Forschung genannt und wie diese möglicherweise in Zukunft gelöst werden.

Kapitel 2

Angriffe gegen Machine Learning Anwendungen

Neben den allgemeingültigen Risiken gegen die Informationssicherheit gibt es eine Reihe spezifischer Risiken von Machine Learning Anwendungen. Im Folgenden werden typische Angriffe gegen Machine Learning Modelle betrachtet und analysiert. Der Fokus liegt dabei auf Angriffen, welche besonders das Schutzziel Vertraulichkeit bedrohen.

2.1 De-Anonymisierung und Re-Identifikation

Für Machine Learning Anwendungen werden, je nach Komplexität der Aufgabe, eine Vielzahl an Daten benötigt. Durch Datenlecks können diese, oftmals private, Daten an die Öffentlichkeit oder in die Hände eines Angreifers gelangen. Ein Beispiel hierfür wäre das Datenleck von Facebook, bei welchem 50 Millionen Nutzerprofile von dem Datenanalyse-Unternehmen Cambridge Analytica ausgelesen worden sind. Diese wurden genutzt, um die US-Wahl 2016 zu beeinflussen [3].

Allerdings kommt es auch vor, dass Unternehmen freiwillig Daten veröffentlichen. Netflix veröffentlichte 2006 einen Datensatz, welcher Filmbewertungen von knapp 500.000 Nutzern enthält [6]. Um nicht absichtlich private Daten zu veröffentlichen, war der Datensatz anonymisiert. Neben einem Wettbewerb steht dieser Datensatz zu Forschungszwecken öffentlich zur Verfügung. Narayanan und Shmatikov [7] zeigen jedoch, dass die Anonymisierung des Datensatzes nicht ausreichend war, um private Informationen zu schützen. Die Bewertungen der anonymisierten Benutzer, wurden mit den Bewertungen der öffentlichen Filmdatenbank IMDb abgeglichen. Dabei genügt es, wenn Präferenzen in Korrelation gesetzt werden können, die genauen Wert sind nicht notwendig. Narayanan und Shmatikov [7] beschreiben, dass sogar politische oder religiöse Informationen herausgefunden werden können. Hierzu werden beispielsweise positive Bewertungen von religiösen Dokumentationsfilmen, die privat auf Netflix abgegeben werden, werden öffentlichen Profilen auf IMDb zugeordnet.

2.2 Property Inference Attacke

Bei der sogenannten Property Inference Attacke versucht ein Angreifer, bestimmte Eigenschaften über die Daten eines Modells herauszufinden, welche nur indirekt von einem Modell gelernt wurden und auch nicht bewusst veröffentlicht wurden [8].

Ateniese et al. [8] zeigten erstmals, wie so ein Angriff bei Machine Learning Modellen ,wie einer Support Vektor Maschine, funktionieren kann. Es wird gezeigt, dass es möglich ist, herauszufinden, ob das angegriffene Modell eine bestimmte Eigenschaft der Daten gelernt hat. Um dies zu erreichen, konstruiert der Angreifer mehrere Datensätze, in denen die zu untersuchende Eigenschaft vorhanden ist oder nicht. Anschließend werden diese Datensätze genutzt, um verschiedene Modelle zu trainieren, welche die gleiche Architektur wie das angegriffene Modell nachbilden. Der Schlüssel dieser Methode ist es, nun einen Meta-Klassifikator mit diesen Modellen zu trainieren. Bei Meta-Klassifikatoren handelt es sich um Modelle, welche aus anderen Modellen lernen. Konkret werden hierbei die Parameter der Modelle als Input genutzt, um vorherzusagen, ob die Trainingsdaten die zu untersuchende Eigenschaft besitzen. Dies ist möglich, da alle Modelle, das angegriffene Modell und die vom Angreifer trainierten Modelle, die gleiche Architektur haben und dadurch auch zum gleichen Format transformiert werden können. Ateniese et al. konnten mit diesem Angriff zeigen, dass es möglich ist, herauszufinden, ob ein Sprachmodell mit Daten der Eigenschaft "Sprecher mit indischem Dialekt "trainiert wurde.

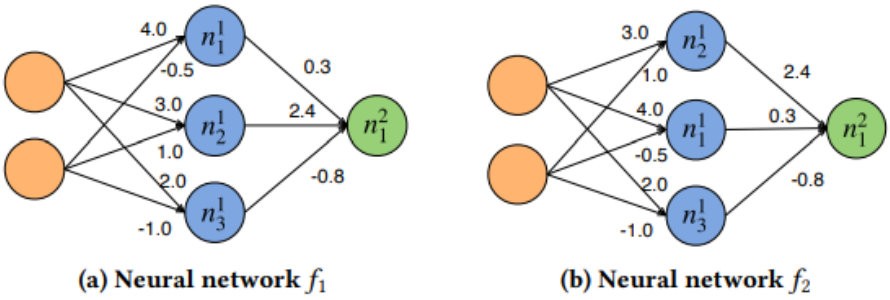

Abbildung 2.1: Permutation eines Neuronalen Netzes [9]

Die Methode von Ateniese et al. [8] ist auf diverse Machine Learning Modelle wie Hidden Markov Modelle oder Support Vektor Maschinen ausgelegt, weshalb diese bei Neuronalen Netzen nicht sonderlich gut funktioniert. Ganju et al. [9] passten den Angriff auf Neuronale Netze an, indem zwei geeignete Repräsentation für diese Art von Modellen genutzt werden kann. Die erste Repräsentation eines Neuronalen Netz basiert auf der Eigenschaft von Neuronalen Netzen, dass die Reihenfolge der Neuronen vertauscht werden kann. Abbildung 2.1 zeigt zwei Neuronale Netze, bei denen die Reihenfolge der Knoten in der ersten Hidden Layer vertauscht ist, jedoch das Modell die gleiche Funktion berechnet. Diese Eigenschaft

kann nun genutzt werden, um jede Schicht zu sortieren und dadurch eine einheitliche Matrix für Permutationen des gleichen Modells zu erhalten.

Die zweite Repräsentation beruht darauf, die Schichten eines Neuronalen Netzes nicht als Vektor darzustellen. Im Gegensatz zu einem Vektor hat ein Set keine feste Reihenfolge oder Ordnung, sondern ist lediglich eine Menge von Objekten, bzw. hier von Knoten. Dies sorgt ebenfalls dafür, dass Permutationen des gleichen Modells, in das gleiche Format übertragen werden können. Ganju et al. [9] zeigen anhand des MNIST Datensatzes, dass beide Repräsentationsformen die Accuracy des Meta-Klassifikators erhöhen können.

Gopinath et al. [10] zeigen, dass viele Informationen eines Neuronalen Netzes in der Aktivierung der Neuronen steckt. Dabei reicht es, einen Wert in **on** und **off** zu unterteilen, wobei on einem Wert > 0 entspricht. Gopinath et al. [10] nutzten diese Darstellung für eine gutwillige Informationsgewinnung über ein Modell, jedoch könnte die Darstellung auch für Property Inference Angriffe genutzt werden.

2.3 Model Inversion Attacke

Bei der Model Inversion Attacke, versucht ein Angreifer, durch bestimmte Eingaben in das Modell, Rückschlüsse zu den Trainingsdaten zu ziehen. Dies kann soweit führen, dass einzelne Datenpunkte nachgebildet werden können [11].

Fredrikson et al. [11] zeigen anhand eines Gesichterkennungsmodells, dass es möglich ist, das Bild einer Person zu rekonstruieren. Diese Person muss lediglich von dem Modell klassifiziert werden können, folglich also auch in den Trainingsdaten vorhanden sein. Abbildung 2.2 zeigt, wie sehr das rekonstruierte Bild (links) dem Originalbild (rechts) ähnelt.

Abbildung 2.2: Rekonstruktion eines Bildes [11]

Der Angriff, auch Reconstruction Attacke genannt, wird durch einen generativen, iterativen Algorithmus durchgeführt. Zu Beginn wird ein Bild als Startbild gesetzt, welches jedem Pixel den Wert 0 zuweist. In jedem Schritt des Algorithmus wird zuerst der Wert einer

Verlustfunktion bestimmt. Der Wert dieser ist, sofern das Modell nicht mehr Details angibt, lediglich der Confidence Score, dass es sich bei dem eingegebenen Bild nicht um das gesuchte Label handelt. Konkret bedeutet dies, wenn das Bild bereits die zu rekonstruierende Person zeigt, der Confidence Score des Labels der Person nahe 1 ist und damit unsere Verlustfunktion nahe 0. Anschließend wird das Bild des vorherigen Durchlaufs, durch einen Autoencoder geschickt, welcher ein Bild für den nächsten Schritt des Algorithmus erzeugt. Bei diesem Autoencoder handelt es sich um ein Neuronales Netz, welches einen Input (hier ein Bild) in einen Vektor mit niedrigerem Rang transformiert und anschließend wieder in einen Vektor mit dem gleichen Rang wie den Input transformiert. Input und Output eines Autoencoders haben somit den gleichen Rang, hier wird also ein Bild zu einem anderen Bild transformiert. Zum Training des Autoencoders werden Bilder genutzt, wobei der Input auch gleich dem Output entspricht. Somit lernt ein Autoencoder, aus einem Bild das gleiche Bild zu erzeugen, jedoch mit der Einschränkung, dass der Vektor innerhalb des Modells einen niedrigen Rang hat. Dieser Algorithmus läuft so lange, bis die maximal konfigurierte Anzahl an Schritten erreicht wurde, die Verlustfunktion einen bestimmten Wert überschritten hat oder sich eine definierte Anzahl an Schritten lang nichts verbessert hat.

Zhang et al. [12] erweitern diesen Angriff, indem die Qualität des generativen Modells (Autoencoder) verbessert wird. Zum einen wird der Autoencoder so erweitert, dass zusätzliche Informationen mitgegeben werden können. Diese zusätzlichen Informationen sind beispielsweise verschwommene oder zensierte Bilder. Eine weitere Verbesserung besteht darin, zusätzlich ein Modell zu nutzen, welches reale Bilder von synthetischen Bildern unterscheiden soll. Dieses Konstrukt entspricht dem Diskriminator eines Generative Adversarial Networks [13]. Mithilfe des Diskriminators, kann der Autoencoder verbessert werden, wodurch die Qualität der einzelnen Bilder erhöht wird und folglich auch die Qualität des rekonstruierten Bildes steigt.

He et al. [14] zeigen eine White Box Version des Angriffs. White Box bedeutet, dass das Modell vollumfänglich in den Händen des Angreifers ist. Dies ist der Fall, wenn ein Modell öffentlich geteilt wird (die Trainingsdaten jedoch nicht). Die Fähigkeit, das Modell vollumfänglich zu nutzen, erlaubt es, das zu rekonstruierende Bild mittels Backpropagation anzupassen. Dazu wird initial ein Bild genutzt, bei jedem jeder Pixel auf einen einheitlichen Farbwert, z. B. 0.5, gesetzt wird. Dieses Bild wird nun durch das Modell interferiert, und der Wert der Verlustfunktion wird backpropagiert. Alle Gewichte und Bias des Modells werden dabei unverändert gelassen, jedoch werden die Pixel des Bildes Gradienten bestimmt und mit einer konfigurierten Lernrate angepasst. Dieser Vorgang wird anschließend so lange wiederholt, bis eine maximale Anzahl an Iterationen durchgelaufen ist.

2.4 Membership Inference Angriff

Bei der Membership Inference Attacke versucht ein Angreifer herauszufinden, ob ein Datenpunkt Bestandteil des Trainingsdatensatzes eines Modells ist. Dies bedroht die Vertraulichkeit, da beispielsweise herausgefunden werden kann, ob eine bestimmte Person Teil eines Trainingsdatensatzes für die Diagnose einer Krankheit ist und folglich auch mit der entsprechenden Krankheit diagnostiziert ist [15].

Shorki et al. [15] führen eine Membership Inference Attacke durch, indem eine Reihe Modelle trainiert wird, die ähnlich dem Angriffsziel-Modell sind. Diese Modelle werden auch Shadow Modelle genannt. Ähnlich bedeutet hier, dass sowohl die Funktion der Modelle, als auch der Trainingsdatensatz, mit dem angegriffenen Modell vergleichbar sind. Einige dieser trainierten Modelle enthalten den zu untersuchenden Datenpunkt im Trainingsdatensatz, andere hingegen nicht. Nun wird ein binärer Meta-Klassifikator trainiert (analog zu der Property Inference Attacke in Kapitel 2.2), welcher anhand der Vorhersagen der Shadow Modelle, Label und Confidence Score, lernt, ob ein Datensatz im Training des entsprechenden Modells genutzt wurde. Wird in diesen Meta-Klassifikator die Vorhersage des angegriffenen Modells als Input genutzt, so erhält man die Antwort, ob der Datenpunkt für das Modelltraining genutzt wurde oder nicht. Laut Shorki et al. [15] funktioniert dieser Angriff, da ähnliche Modelle, die mit einem ähnlichen Datensatz trainiert wurden, sich auch ähnlich verhalten. Somit kann bei den selbst trainierten Modellen, welche den Datensatz enthalten, ein Muster gelernt werden, welches auch auf andere Modelle anwendbar ist. Overfitting und eine komplexe Modellarchitektur erhöhen die Wahrscheinlichkeit eines erfolgreichen Angriffs.

Carlini et al. [16] zeigen eine Alternative des Angriffs, bei welcher kein Meta-Klassifikator genutzt wurde. Die Shadow Modelle werden analog zu [15] trainiert. Anstatt mit den Vorhersagen dieser Modelle nun einen Meta-Klassifikator zu trainieren, werden zwei Gaußverteilungen gebildet, jeweils über die Confidence Scores der Modelle, wo der Datenpunkt im Training enthalten war oder nicht. Mittels eines Likelihood-Quotienten-Tests wird anschließend vorhergesagt, in welcher Verteilung der Confidence Score des angegriffenen Modells wahrscheinlicher liegt. Ein Vorteil dieser Methode liegt darin, dass weniger Shadow Modelle trainiert werden müssen, da bereits mit relativ wenig Werten eine Gaußverteilung modelliert werden kann.

Eine Voraussetzung bei Shorki et al. [15] ist es, dass der Confidence Score mit ausgegeben wird. Choquette-Choo et al. [17] wandeln den Angriff ab, sodass dieser Score nicht mehr benötigt wird. Der Angriff funktioniert simultan zu [15], jedoch wird nicht nur der Datenpunkt selber in die Modelle als Input gegeben, sondern auch Abwandlung davon. Diese Abwandlungen könnte das Hinzufügen von zufälligem Rauschen sein, oder bei Bilddateien beispielsweise Rotation oder Translation. Die Hypothese der Autoren ist, dass das Modell

bei Datenpunkten, die im Training genutzt wurden, robuster gegenüber diesen Abwandlungen ist und dennoch den Datenpunkt korrekt klassifiziert. Zusätzlich könnten Abwandlungen der Daten über einen Data Augmentation Schritt auch direkt vom Modell gelernt worden sein.

2.5 Data Extraction Attacke

Bei der Data Extraction Attacke versucht ein Angreifer, Informationen eines Modells zu extrahieren, die gelernt wurden, obwohl dies (oftmals) nicht der Fall sein sollte [18]. Der Angriff unterscheidet sich von der Model Inversion Attacke (Kapitel 2.3) und von der Property Inference Attacke (Kapitel 2.2), indem Daten direkt aus dem Modell extrahiert werden und nicht anhand der Vorhersage des Modells nachgebildet werden.

Carlini et al. [18] beschrieben, dass konkrete Zeichenketten oder Werte, wie eine Kreditkartennummer oder eine Sozialversicherungsnummer, in einem Sprachmodell gelernt werden. Um dies zu evaluieren, wurde ein Sprachmodell mit dem Penn Treebank Datensatz trainiert, welcher ca. 5MB groß ist. Zusätzlich wurde ein Satz eingefügt, welcher mit *"My social security number is "* beginnt und anschließend eine Zahlenfolge beinhaltet. Die Funktionalität des Modells liegt darin, das nächste Wort oder Zeichen vorherzusagen, wenn eine Zeichenkette eingegeben wird. Anzumerken ist hierbei noch, dass dieses Modell signifikant kleiner als 5 MB ist, was folglich bedeutet, dass nicht alle Trainingsdaten in dem Modell gespeichert sein können. Die Experimente von Carlini et al. [18] zeigen, dass dieses Modell die Zahlenfolge ungewollt gelernt hatte als mögliche Vorhersage ausgibt, wenn der oben genannte Satz als Input genutzt wird.

In einem anderen Forschungsprojekt, ebenfalls unter der Leitung von Carlini, [19] zeigen die Autoren eine Data Extraction Attacke am Beispiel des Sprachmodells GPT-2. Dabei handelt es sich um ein Sprachmodell des Unternehmens OpenAI, welches der Vorgänger von ChatGPT (siehe Kapitel 1) ist und Open Source zur Verfügung steht. Obwohl voller Zugriff auf das Modell besteht, wird lediglich die Ausgabe des Modells betrachtet. Folglich bedeutet dies, dass der Angriff auf jedes Modell anwendbar wäre. Zur Durchführung des Angriffs wird lediglich ein Starttoken in das Modell eingegeben und anschließend vielfach das vorgeschlagene Folgewort gesammelt. Wird dies lang genug gemacht, erhält man eine lange Tokenabfolgen, also quasi Sätze, die vom Modell gelernt wurden. Dabei kann es sich um öffentliche Texte handeln, wie beispielsweise der Text der MIT Open Source Lizenz, aber auch private Daten wie Email-Adressen sind vorhanden. Diese Variation des Angriffs kann in gewissem Maße funktionieren, liefert jedoch oftmals gleiche Wortabfolgen und hat auch eine hohe False-Positive Rate. Carlini et al. [19] variierten deshalb die Methodik, wie die Tokenabfolge gesammelt wird. Bevor GPT-2 das wahrscheinlichste Folgewort vorschlägt, werden die Wahrscheinlichkeiten in den Wertebereich (0,1) transformiert und so skaliert,

dass diese Werte addiert 1 ergeben. Wird der Softmax Funktion ein Hyperparameter namens Temperatur > 1 mitgegeben, wird das Modell unsicherer und erhöht dadurch die Diversität der Vorhersagen des Modells. Neben dieser Temperatur wird eine zweite Verbesserung vorgeschlagen. Anstatt nur einen Starttoken zu nutzen, werden die ersten Wörter von verschiedenen, öffentlichen Datenquellen genutzt. Mit diesen zwei Verbesserungen konnten mehr unterschiedliche Arten von Texten, die das Modell gelernt hat, extrahiert werden. Neben Newsartikeln oder Forumsbeiträgen, befanden sich auch Kontaktdaten einiger Privatpersonen in diesen Tokenabfolgen.

2.6 Poisoning Attacke

Bei der sogenannten Poisoning Attacke werden manipulierte Datensätze in den Trainingsdatensatz eines Modells injiziert, wodurch das Modell schlechtere oder sogar falsche Vorhersagen trifft. Ursprünglich ist diese Art des Angriffs recht populär bei Support Vektor Maschinen. Biggio et al. [20] zeigt, dass einige modifizierte Datenpunkte in der Nähe der Entscheidungsgrenze genügen, um die gelernte Funktion deutlich negativ zu beeinflussen.

Yang et al. [21] zeigen ein Verfahren, bei dem eine Poisoning Attacke auf Neuronalen Netzen angewendet wird. Ziel hierbei ist es, die Daten mit einem falschen Label so zu wählen, dass der Wert der Verlustfunktion möglichst groß ist. Dadurch werden die Gradienten größer, was auch bedeutet, dass die negative Beeinflussung des infizierten Datenpunktes größer wird. Um diese Daten zu erzeugen, nutzen Yang et al. [21] einen Autoencoder, der Daten so transformiert, dass diese vom Modell als echte Daten erkannt werden, jedoch aufgrund der falschen Labels, das Modell möglichst stark negativ beeinflussen. Dieser Autoencoder wurde trainiert, indem die Verlustfunktion des angegriffenen Modells auch durch den Autoencoder backpropagiert wurde.

Guo und Liu [22] nutzen einen Ansatz, bei welchem der Angreifer keinen Zugriff auf die Gradienten des angegriffenen Modells braucht. Stattdessen wird ein vortrainiertes Modell genutzt, welches ähnlich zu dem angegriffenen Modell ist. Da diverse Modellarchitekturen Open Source sind, finden sich auch einige vortrainierte Varianten von diesen im Internet. Bei Bildklassifikation lässt sich beispielsweise ein vortrainiertes YOLO Modell nutzen. Dieses kann dann genutzt werden, um ein generatives Modell zu trainieren, welches wie bei Yang et al. [21] die Gradienten durch das Generator Modell backpropagieren. Guo und Liu [22] gehen davon aus, dass das angegriffene Modell noch optimiert wurde und deshalb eine bessere Feature Erkennung als die öffentlich vortrainierten Modelle hat. Dies macht den Angriff effektiver, sofern die Modelle nicht zu verschieden sind.

Poisoning Attacken verschlechtern in der Regel lediglich die Performance eines Modells und sorgen für falsche Vorhersagen. Tramèr et al. [23] zeigen jedoch, dass manipulierte Daten

dafür sorgen können, dass andere Angriffe, welche die Vertraulichkeit angreifen, effektiver werden. Durch Ändern des Labels eines Datenpunktes kann dieser gegebenenfalls zu einem Ausreißer transformiert werden. Dadurch passt sich das Modell stärker diesem an, als wenn sich der Datenpunkt in die Messreihe einordnet.

2.7 Angriffe gegen Verteiltes Lernen

Die Größe und Komplexität eines Machine Learning Modells korreliert mit dem Funktionsumfang der zu bewältigen Aufgabe. Dies führt dazu, dass eine große Datenmenge und mehr Rechenleistung benötigt werden. Ist eines dieser beiden Ressourcen knapp, können diese von mehreren Partnern geteilt werden. Das Verteilte Lernen birgt jedoch einige besondere Risiken, welche im Folgenden detaillierter betrachtet werden.

Verteiltes Lernen, auch Federated Learning oder Collaborative Learning genannt, kann unterschiedlich durchgeführt werden. Diverse Parameter, beispielsweise wie oft Systeme ihre Änderungen übermittelt, können konfiguriert werden. Jedoch ist eine wichtige Unterscheidung die Topologie des Systems, welche in Abbildung 2.3 gezeigt werden. Links sieht man ein System, welches einen zentralisierten Server benutzt. Die Clients, dargestellt durch Smartphones, berechnen mit ihren privaten Daten Gradienten, welche dann an einen zentralen Server übermittelt werden. Dort findet die Anpassung des Modells statt. Rechts sieht man einen dezentralen Ansatz, bei dem Clients miteinander vernetzt sind und ihre Updates (z. B. Gradienten) untereinander austauschen [24].

Abbildung 2.3: Verteiltes Lernen Topologien [24]

Melis et al. [25] zeigen, dass Verteiltes Lernen anfällig für Membership Inference Attacken (siehe Kapitel 2.4) und Property Inference Attacken (siehe Kapitel 2.2) sind. Ein Membership Inference Angriff kann durchgeführt werden, indem die Gradienten der Eingabeschicht eines Neuronalen Netzes beobachtet werden. Bei Textmodellen beispielsweise, ist es oftmals der Fall, dass die Worte bzw. Tokens in einen Vektor übertragen werden. Dieser Vektor enthält dabei die Information, welche Knoten der ersten Schicht aktiviert sind. An besagten Stellen der ersten Schicht, sind die Gradienten deshalb ungleich 0. Durch diese Beobachtung kann ein Angreifer also feststellen, welche Wörter in einem Datenpunkt oder einem Batch aus Datenpunkten enthalten sind. Für einen Property Inference Angriff, kann der

Angreifer verschiedene Snapshots des gemeinsam trainierten Modells nutzen. Jeder dieser Snapshots wird zweimal separat weitertrainiert, einmal mit einem Datensatz, welcher die zu untersuchende Eigenschaft enthält und einmal mit einem Datensatz der diese nicht enthält. Die Unterschiede von den ursprünglichen Snapshot-Modellen und den weitertrainierten Modellen sind folglich die Inputs und die Labels sind die Information, ob Daten mit der Eigenschaft oder ohne die Eigenschaft zum Lernen genutzt wurden. Damit kann nun ein Meta-Klassifikator trainiert werden, der vorhersagt, ob ein Modell weitertrainiert wurde, mit Daten, welche die zu untersuchende Eigenschaft enthält. Bei jedem Update des gemeinsam gelernten Modells kann dieser Meta-Klassifikator genutzt werden.

Zhu et al. [26] beschrieben einen Angriff auf Verteilte Systeme, welcher Deep Leakage genannt wird. Der Angriff funktioniert sowohl bei einer zentralisierten als auch einer dezentralisierten Topologie. Beim zentralisierten System muss der Angreifer jedoch Zugriff auf den zentralen Server haben, beim dezentralisierten System reicht es, ein Teilnehmer zu sein. Dies liegt daran, dass die Gradienten (bzw. eine Gradientenmatrix) mit dem Angreifer geteilt werden müssen, damit der Angriff funktioniert. Im Laufe des Trainingsprozesses werden mehrfach Gradienten dem Angreifer übergeben, und für jeden Austausch, können die eingegebenen Trainingsdaten ermittelt werden. Um den Angriff für eine Gradientenmatrix durchzuführen, wird initial ein Eingabevektor generiert. Dieser wird durch das gemeinsam zu trainierende Modell interferiert, der Wert der Verlustfunktion bestimmt und anschließend durch das Modell backpropagiert. Anstatt jedoch die Gewichte der Knoten des Modells anzupassen, werden nur die Werte des Eingabevektors iterativ (multipliziert mit einer festgelegten Lernrate) angepasst, sodass sich die Gradienten des Eingabevektors der geteilten Gradientenmatrix annähern. Durch diese Angleichung, wird der ursprünglich initial gesetzte Eingabevektor immer ähnlicher zu den originalen Trainingsdaten, die ein anderer Nutzer des verteilten Systems zum Training genutzt hat. Mit dieser Attacke, konnten die Autoren zeigen, dass die rekonstruierten Bilder nahezu (bis auf einige Pixel) identisch zu den Originalbildern sind. Abbildung 2.4 zeigt einer Gegenüberstellung der Bilder. Die Bilder können ebenfalls rekonstruiert werden, wenn mehrere Bilder in Batches zum Training genutzt werden.

Zhao et al. [27] optimierten den Deep Leakage Angriff, indem ein zusätzlicher Schritt in den Algorithmus eingefügt wird. Dieser beinhaltet, dass zuerst das Label des Datenpunktes herausgefunden wird, von dem die Gradienten stammen. Dies kann dadurch ermittelt werden, dass die Kostenfunktion eines Outputs bei der richtigen Klasse den Wertebereich (-1,0) annimmt und bei den falschen Klassen den Wertebereich (0,1), sofern keine Quadrierung stattfindet. Der initiale Eingabevektor hat nun von Beginn an das richtige Label. Dadurch werden weniger Iterationen benötigt, um Daten zu rekonstruieren.

Hitaj et al. [28] zeigen einen alternativen Angriff. Bei diesem Szenario einigen sich die Trainingsteilnehmer darauf, von welchen Klassen sie jeweils Daten haben und entsprechend auch

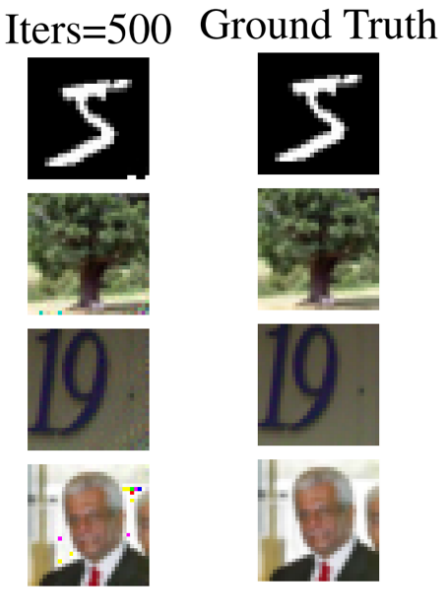

Abbildung 2.4: Rekonstruktion durch Deep Leakage [26]

welche Sie davon im gemeinsamen Modell trainieren. Klassen können sich unter den Teilnehmern überlappen, sodass mehrere Teilnehmer diese Klasse trainieren können. Jedoch braucht der Angreifer eine Klasse, die außer ihm niemand hat. Diese muss nicht wirklich in den Daten vorkommen, sondern es geht darum, dass kein anderer Teilnehmer die Klassifikation dieser Klasse beeinflusst. Zur Durchführung des Angriffs, wird das Modell wie geplant trainiert, bis eine gewisse Güte erreicht ist. Der Angreifer versucht nun, Daten einer Klasse zu rekonstruieren, die nur von anderen Teilnehmern bespielt wurde. Dazu wird ein Generative Adversarial Network [13] genutzt. Dabei handelt es sich um eine Architektur, welche sich aus zwei Modellen, einem Generator und einem Discriminator, zusammensetzt. Der Discriminator entscheidet dabei, ob die Daten, die vom Generator erzeugt werden, echt oder unecht sind. In diesem Angriff wird als Discriminator das gemeinsam gelernte Modell genutzt, welches vorhersagt, ob die Daten Teil der angegriffenen Klasse sind oder nicht. Der Generator wird vom Angreifer trainiert, indem er das gemeinsame Modell auch durch den Generator backpropagiert. Diese erzeugten Daten, kann der Angreifer der Klasse zuordnen, die außer ihm niemand trainiert. Daduruch werden andere Teilnehmer gedrängt, mehr oder öfters Daten für die angegriffene Klasse preis zu geben, da das gemeinsame Modell nicht mehr zwischen der angegriffenen Klasse und der zugeordneten Klasse des Angreifers unterscheiden kann. Anschließend kann der Angreifer erneut den Generator verbessern. Anzumerken ist hierbei noch, dass der Generator nicht die Trainingsdaten im Detail rekonstruiert, sondern nur Daten erzeugt, welche der gleichen Klasse zugeordnet werden können. Die Autoren zeigen jedoch, dass bei Bildern eine erkennbare Ähnlichkeit vorhanden ist.

Kapitel 3

Methoden zur Sicherung der Vertraulichkeit

Neben den beschriebenen Angriffen aus Kapitel 2 gibt es eine Vielzahl an Methoden, diese abzuschwächen oder sogar ganz zu unterbinden. Dieses Kapitel stellt zuerst eine typische Pipeline zum Training eines Neuronalen Netzes vor und teilt diese in 3 Phasen ein. Phase 1 ist dabei die Aufbereitung des Datensatzes, also jegliche Verarbeitung der Daten, die vor dem eigentlichen Training stattfinden. Anschließend folgt Phase 2, welche das Training des Modells umfasst. Das Ende der Pipeline, Phase 3, beinhaltet das Deployment so wie den Betrieb des Modells. Die Gegenmaßnahmen werden jeweils der entsprechenden Phase untergeordnet.

3.1 Übersicht Machine Learning Pipeline

Abbildung 3.1 zeigt den Aufbau der typischen Pipeline, um ein Neuronales Netz zu trainieren. Diese wird im Folgenden genauer beschrieben. Schritt 1 ist die Vorverarbeitung der Daten. Die Daten können dabei aus einer Datenbank oder einem Filesystem stammen, die im Voraus gesammelt wurden. Die Vorverarbeitung der Daten ist recht individuell und kann je nach Anwendungsfall variieren. Typische Handlungen sind:

- Vereinheitlichung des Datenformats
- Fehler und starke Ausreißer werden entfernt
- Normalisierung der Daten
- Erstellung neuer Daten durch Transformationen
- Kodierung von (kategorialen) Variablen und Labels
- Aufteilung in Trainings-, Validierungs- und Testdatensätze

Die Abbildung 3.1 zeigt stellt diesen Schritt dar, bei dem mehrere Datenquellen verbunden werden und nur ein Dokument übrig ist. Dieses eine Dokument soll zeigen, dass nur die wichtigen Informationen der Daten erhalten bleiben, jedoch kann es in der Praxis sein, dass die Datenmenge beim Aufbereiten der Daten größer wird. Kapitel 3.3 stellt verschiedene Methoden vor, die bei der Vorverarbeitung der Daten angewendet werden können, um die Vertraulichkeit zu sichern.

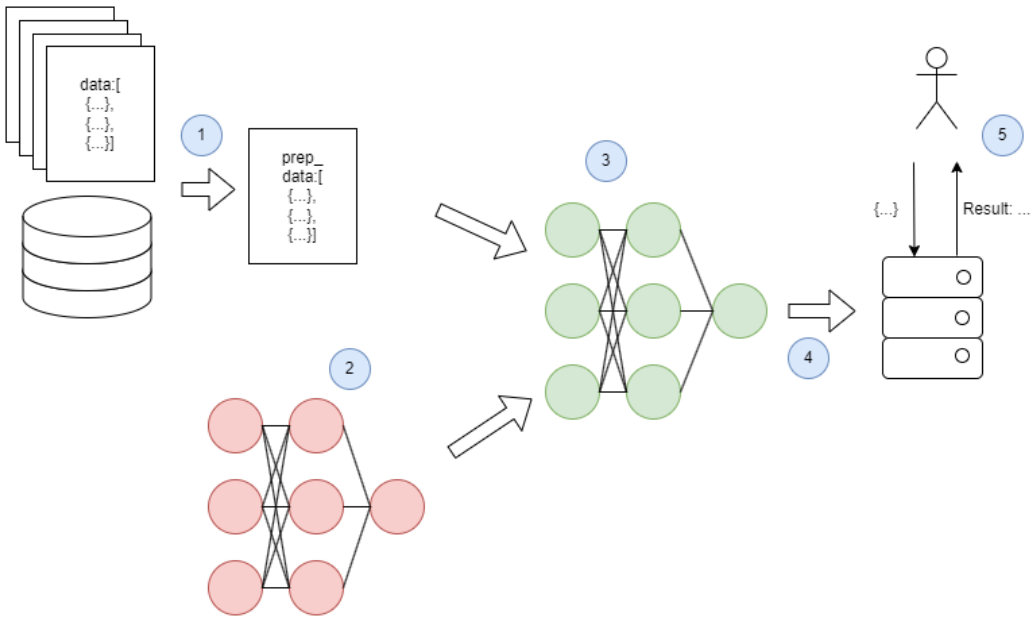

Abbildung 3.1: Training eines Neuronalen Netzes

Schritt 2 umfasst die Architektur eines Modells. Je nach Anwendungsfall und Komplexität der Aufgabe, werden unterschiedliche Konfigurationen der einzelnen Schichten vorgenommen. In der Abbildung wird dies durch ein rotes Modell angezeigt, da Gewichte des Modell zufällig (oder anhand einer Verteilung) initialisiert sind, und noch keine sinnvolle Vorhersage getroffen werden kann. Schritt 3 ist der tatsächliche Trainingsvorgang, bei dem das Modell lernt. Grafisch wird dies durch den Verbund des Dokuments und des untrainierten, roten Modells dargestellt. Es entsteht ein grünes Modell, welches in der Lage ist, sinnvolle Vorhersagen zu erzeugen. In der Praxis gibt es hier einige Iterationen des Training, welche zusätzliche Vorgänge, wie beispielsweise eine Validierung, enthalten. Kapitel 3.3 widmet sich Methoden, die während der Training, welche aus Schritt 2 und 3 besteht, angewendet werden.

Schritt 4 und 5 beschreiben das Deployment und den Betrieb des Modells. Dieses soll für vorgesehene Anwender erreichbar sein. Je nach Anwendungsfall kann die Systemarchitektur unterschiedlich aussehen. Einige Modelle werden mit Trainingscode auf öffentlichen Plattformen, wie HuggingFace[1], geteilt, wohingegen andere Modelle, wie ChatGPT[2], nur über eine spezielle Oberfläche erreichbar sind. In Abbildung 3.1 zeigt deshalb lediglich einen Anwender, der direkt Request an das Modell schickt. In Kapitel 3.5 werden Maßnahmen besprochen, die in dieser Phase die Vertraulichkeit sichern. Dabei kann es sich um Transformationen des Modells vor dem Deployment handeln, oder um Mechanismen, die Anfragen und Antworten des Modells abwandeln.

[1] https://huggingface.co/models
[2] https://chat.openai.com/

3.2 Aufbereitung des Datensatzes

Das folgende Kapitel zeigt diverse Methoden, wie Daten vor dem eigentlichen Training angepasst werden können, damit Angriffe auf die Vertraulichkeit abgeschwächt werden. Die Kategorisierung der Techniken erfolgt anhand der Art und Weise, wie die Daten abgeändert werden.

3.2.1 Anonymisierung

Bei der Anonymisierung von Daten geht es darum, identifizierende Eigenschaften zu entfernen oder unkenntbar zu machen. Dabei soll dennoch ein gewisser Nutzen erhalten bleiben.

Sweeney [29] stellt eine Methode der Anonymisierung namens k-Anonymität (im Englischen **k**-Anonymity) vor. Die Methode wird im folgenden mittels eines Auszugs des Titanic Datensatzes [30] erläutert. Tabelle 3.1 zeigt einen Auszug aus diesem Datensatz. Die hier ausgewählten Spalten enthalten beschreibende Merkmale einer Person, sowie Informationen über die Reise auf der Titanic. Zur Verdeutlichung gehen wir davon aus, dass es sich bei dem Einstiegsort um eine private Information handelt, die den Wohnort verraten könnte.

Name	Geschlecht	Alter	Buchungsklasse	Einstiegsort
Owen Harris Braund	m	22	3	S
Florence Briggs Thayer	w	38	1	C
Laina Heikkinen	w	26	3	S
Lily May Peel	w	35	1	S
William Henry Allen	m	35	3	S
Anna McGowan	w	15	3	Q
James Moran	m	30	3	Q
Adele Achem Nasser	w	14	2	C
Don Manuel Uruchurtu	m	40	1	C
Elizabeth Anne Wilkinson	w	29	2	S
Henry Birkhardt Harris	m	45	1	S

Tabelle 3.1: Nicht-Anonymisierter Titanic Datensatz [30]

Bei k-Anonymität werden die Attribute in 3 separate Klassen eingeteilt: Identifikatoren, Quasi-Identifikatoren und sensible Attribute. Bei diesem Beispiel wäre der Name der einzige Identifikator, da über diesen eine Person eindeutig zugeordnet werden kann. Quasi-Identifikatoren sind alle Variablen, welche Information über einen Datenpunkt preisgeben, aber nicht direkt auf diesen schließen lassen. Um mit Quasi-Identifikatoren einen Datenpunkt zu identifizieren, braucht es in der Regel mehr Informationen, beispielsweise einen Datensatz. In diesem Beispiel könnte also jede Spalte ein Quasi-Identifikator sein. Da der Name bereits ein direkter Identifikator ist, wird dieser anders zugeordnet. Die dritte Klasse sind die sensiblen Attribute, die es zu sichern gilt. Hier gehen wir davon aus, dass der

Einstiegsort eine schützenswerte Information ist. Um k-Anonymität zu erreichen, muss jede Kombination aus Quasi-Identifikatoren mindestens k mal vorkommen, wobei k festgelegt werden kann. Ein größeres k sorgt für mehr Privatsphäre. Um dies zu erreichen, können die Quasi-Identifikatoren gruppiert werden, so kann beispielsweise anstatt des Alters, eine Zahlenbereich als Alter dienen. Tabelle 3.2 zeigt wie diese Gruppierung aussieht.

Identifikator	Quasi-Identifikatoren			sensibles Attribut
Name	Geschlecht	Alter	Buchungsklasse	Einstiegsort
-	m	20 - 35	3	S
-	m	20 - 35	3	S
-	m	20 - 35	3	Q
-	w	15 - 30	3	S
-	w	15 - 30	3	Q
-	w	10 - 30	2	C
-	w	10 - 30	2	S
-	w	30 - 40	1	C
-	w	30 - 40	1	S
-	m	40 - 45	1	C
-	m	40 - 45	1	S

Tabelle 3.2: **k**-Anonymity Titanic Datensat [29][30]

Es ist zu sehen, dass die Identifikatoren, hier nur der Name, entfernt wurden. Geschlecht und Buchungsklasse sind auch unverändert geblieben. Die Gruppierung erfolgte anhand der Quasi-Identifikatoren, wobei das Alter durch einen Zahlenbereich ersetzt wurde. Hier ist anzumerken, dass die Spanne der Altersgruppen unterschiedlich groß ist. Je nachdem wie diese Spannen aufgebaut sind, würden sich die Gruppierungen verändern. k-Anonymität mit k = 2 ist hier erfüllt, da jede Kombination von Quasi-Identifikatoren mindestens 2 mal vorkommt. Dabei ist es auch möglich, dass einzelne Einträge redundant vorkommen. So sind in Tabelle 3.2 die ersten beiden Einträge identisch, obwohl diese von 2 unterschiedlichen Personen stammen.

Machanavajjhala et al. [31] zeigen anhand von 2 Attacken, dass k-Anonymität nicht ausreichend ist. Bei der Homogenitätsattacke kann die Eigenschaft, dass sensible Attribute nicht einzigartig sind, ausgenutzt werden. Tabelle 3.3 zeigt abgewandelt die ersten Zeilen der Tabelle 3.2. Das sensible Attribut, der Einstiegsort, ist jedoch in dieser Quasi-Identifikatoren Kombination identisch. Sollte also eine Person männlich, zwischen 20 und 35 Jahren alt sein und in der Buchungsklasse 3 mitgefahren sein, so kennt man auch den Einstiegsort.

Ein weiteres Problem ist ein Angriff mit Hintergrundwissen, mit dem gewisse sensible Attribute ausgeschlossen werden können oder zumindest unwahrscheinlicher machen könnten. In

Identifikator	Quasi-Identifikatoren			sensibles Attribut
Name	Geschlecht	Alter	Buchungsklasse	Einstiegsort
-	m	20 - 35	3	S
-	m	20 - 35	3	S
-	m	20 - 35	3	S

Tabelle 3.3: Angreifbare Abwandlung von Tabelle 3.2

diesem Beispiel könnte es also sein, dass ein Angreifer weiß wann, ein Passagier zugestiegen ist und an welchem Hafen das Schiff zu dieser Zeit war. Damit können Rückschlüsse auf den Einstiegsort gemacht werden.

Aufgrund dieser beiden Angriffe, schlagen Machanavajjhala et al. [31] eine Erweiterung mit dem Namen l-Diversität (im Englischen **l**-Diversity) vor. Dabei wird k-Anonymität auf die Daten angewendet und zusätzlich eine Bedingung eingeführt. Diese kann sowohl für einen Block, eine einheitliche Kombination der Werte der Quasi-Identifikatoren, als auch für den ganzen Datensatz gelten. Ein Block ist dabei l-divers, wenn die Werte des sensiblen Attributes *"gut repräsentiert"* sind, wobei l eine Zahl zugeorndet werden kann. Ist jeder Block des Datensatzes l-divers, dann ist auch der ganze Datensatz l-divers. Dabei gibt es 3 Grundvarianten laut Machanavajjhala et al., wie *"gut repräsentiert"* definiert werden kann [31]:

- **Unterscheidbare *l*-Diversität:** Bei dieser Variante, hat ein Block **l** unterschiedliche Werte eines sensiblen Attributes. Ein Block ist daher immer mindestens 1-divers, da dies bedeutet, dass das sensible Attribute immer den gleichen Wert annimmt.
- **Entropie *l*-Diversität:** Hier wird die Entropie der sensiblen Attribute eines Blocks berechnet. Dabei ist ein Block l-divers, wenn die Entropie $\geq \log(l)$ ist. Folglich ist 1-Diversität dabei immer gegeben.
- **Rekursive (c,*l*)-Diversität:** Diese Definition besagt, dass das häufigste sensible Attribut eines Blocks, seltener vorkommt, als die Anzahl der restlichen Attribute, multipliziert mit einem konstanten Wert c. Folglich darf kein sensibles Attribut zu oft vorkommen. Ein Block ist dabei (c, l)-divers, wenn $\mathbf{l} - 1$ verschiedene einzigartige Attribute entfernt werden können und die Bedingung immernoch erfüllt ist.

Je nach Datensatz, kann es Sinn ergeben, einige Ausnahmen zu erlauben. So könnte es sein, dass ein Datensatz von einer Variable dominiert wird, die jedoch keine Verletzung der Privatsphäre darstellt. Ein Beispiel der Autoren ist, wenn eine Kardiologie preisgibt, dass die meisten Patienten eine Herzkrankheit haben. Auf der anderen Seite, gibt es Attribute, die besonders geschützt werden sollten.

Sollte ein Datensatz mehrere sensible Attribute besitzen, so muss l-Diversität für jede dieser Attribute gelten. Für diese Überprüfung, werden jeweils alle anderen Spalten, auch die sensiblen Attribute, als Quasi-Identifikator angesehen.

Li et al. [32] zeigen, dass l-Diversität zwei Angriffsflächen bietet. Die erste Angriffsfläche ergibt sich, wenn die Verteilung des sensiblen Attributs sehr stark links- oder rechtsschief ist. Die Autoren zeigen ein Beispiel, bei der das sensible Attribut eine Infektion mit einem bestimmten Virus ist. Dabei sind 99% der Personen gesund und lediglich 1% der Personen infiziert. Die Verteilung des Attributs ist stark schief. Hat jetzt ein Block, der durch **k**-Anonymität entsteht, ein 50% Aufteilung beider Werte, so wäre dieser Block l-divers mit $l = 2$. Kann man jedoch eine Person diesem Block zuordnen, so wäre dies ein Informationsgewinn, da besagte Person ein überdurchschnittliches Risiko der Infektion besitzt. Die zweite Angriffsfläche entsteht dadurch, dass l-Diversität nicht berücksichtigt, ob die Werte des Attributes eine ähnliche Bedeutung haben. Bei einem Krankheitsbeispiel könnten die Werte alle unterschiedliche Krankheiten annehmen, die jedoch das gleiche Körperteil betreffen. Diese Angriffsfläche ähnelt der Homogenitätsattacke gegen k-Anonymität, bloß dass hier zusätzlich Werte semantisch verbunden werden können.

Aufgrund dieser beiden Angriffsflächen stellen Li et al. [32] ein neues Maß an Sicherheit vor: t-Nähe (im Englischen **t**-Closseness). Ziel dieses Maßes ist es, zu zeigen, dass die Verteilung eines sensiblen Attributes in einem einzelnen Block ähnlich zu der Verteilung des gleichen Attributes im gesamten Datensatz ist. Der Unterschied zwischen den beiden Verteilungen soll kleiner als ein Grenzwert t sein. Die Autoren prüfen verschiedene Verfahren der Distanzmessung der Verteilungen und favorisieren die sogenannte Earth Mover Distanz. Dabei handelt es sich um eine Metrik zweier Verteilungen, welche die minimale Arbeit berechnet, die nötig ist, um eine Verteilung zu der anderen Verteilung zu transformieren, indem Werte innerhalb der Verteilung verschoben werden. Die Metrik liegt immer im Wertebereich (0,1) wodurch diese auch vergleichbar ist. Ein Wert nahe 0 ist dabei besser. Mathematisch gesehen, handelt es sich um ein Optimierungsproblem, jedoch gehen die Autoren auf 2 unterschiedliche Arten von Attributen ein, numerische und kategoriale, um zu zeigen, wie die Earth Mover Distanz berechnet wird. Um die Distanz für numerische Werte berechnet zu werden, müssen diese erstmal sortiert werden. Sofern es sich um eine ungleiche Anzahl an Werten handelt, können die Werte mehrfach genutzt werden. Anschließend wird die durchschnittliche, normalisierte Differenz zwischen den Werten an gleicher Stelle beider sortierten Verteilungen berechnet. Im Folgenden wird eine Beispielrechnung exerziert, welches ein sensibles Attribut, Stundenlohn in Euro, darstellt. Verteilung 1 ist dabei das sortierte Gehalt eines Blockes nach **k**-Anonymität und Verteilung 2 das Gehalt des gesamten Datensatzes:

Verteilung 1 = {20, 30, 40}
Verteilung 2 = {20, 25, 25, 30, 35, 35, 35, 40, 40}

Da Verteilung 2 dreimal so viele Elemente enthält wie Verteilung 1, wird jedes Element dreimal genutzt. Dadurch erhält man:

Verteilung 1' = {20, 20, 20, 30, 30, 30, 40, 40, 40}

Die größte Differenz ist 40 – 20 = 20, somit wird der Betrag jeder Differenz durch 20 dividiert, dass diese jeweils im Wertebereich (0,1) liegen. Werden jetzt die einzelnen Wertepaare verglichen, so ergibt sich folgend Distanz:

$$(20-20)+(20-25)+(20-25)+(30-30)+(30-35)+(30-35)+(40-35)+(40-35)+(40-40)=-10$$

Der durchschnittliche, normalisierte Wert dieser Distanz, ist die gesuchte Earth Mover Distanz:

$$1/9 \times |-10| \div /20 = 0,056$$

Damit hat dieser Block eine 0,056-Nähe, was bedeutet, wenn man einer Person diesem Block zuordnet könnte, ist dennoch kaum Informationsgewinnung möglich.

Bei kategorialen Werten ist es schwieriger, eine Differenz zu bilden. Es gibt die Möglichkeit den Wert 1 zuzuweisen, wenn die beiden Kategorien unterschiedlich sind und den Wert 0, sofern beide gleich sind. Dies würde jedoch bedeuten, dass semantische Ähnlichkeiten der Werte nicht berücksichtigt werden. Eine Alternative wäre es, alle möglichen Werte semantisch zu in einer Art Baumstruktur zu gliedern. Bei Krankheiten wäre beispielsweise die Wurzel *"Krankheit"*, die Nachfolger wären dann gewisse Systeme des Körpers wie beispielsweise *"Herz-Kreislaufsystem"* und *"Verdauungssystem"*. Die Distanz ist nun die Anzahl der Schritte, die benötigt wird, um die Werte zu verbinden. Zwei unterschiedliche Herzkrankheiten sind über einen Schritt mittels *"Herz-Kreislaufsystem"* verbunden, wohingegen eine Herzkrankheit und eine Darmkrankheit über 2 Schritte mittels der Wurzel *"Krankheit"* verbunden wären.

3.2.2 Differential Privacy

Differential Privacy ist eine Technik, welche 2006 von Cynthia Dwork [33] vorgestellt wurde. Ziel dabei ist es, Zugriff auf einen Datensatz zu ermöglichen, der sowohl nützliche Erkenntnisse zulässt, als auch die Privatsphäre eines einzelnen Datenpunktes schützt.

Diese beiden Ziele sind laut Dwork [33] erfüllt, wenn Anfragen auf zwei Datensätze, die sich in höchstens einem Datenpunkt unterscheiden, keine signifikanten Unterschiede aufweise. Um dies zu erreichen, wird den Anfragen ein zufälliges Rauschen hinzugefügt. Dadurch handelt es sich bei Abfragen um randomisierte Funktionen. Die Größe des Unterschied der gleichen Abfrage auf zwei Datensätze, die sich in höchstens einem Eintrag unterscheiden, kann durch einen Wert ϵ dargestellt werden.

Formal lautet die Definition von ϵ-Differential Privacy wie folgt [33]:
Eine randomisierte Funktion M, welche einen Datensatz D auf einen Wertebereich R abbildet, weist ϵ-Differential Privacy auf, wenn für alle Datensätze D_1 und D_2 die sich in höchstens einem Datenpunkt unterscheiden, gilt:

$$Pr[M(D_1) \in R] \leq e^{\epsilon} \times Pr[\kappa(D_2) \in R] \tag{3.1}$$

Dwork und Roth [34] fügten der Definition noch einen Parameter δ hinzu, welcher erlaubt, dass die Voraussetzungen zu einem definierten Grad verletzt werden können. Die damit angepasste Definition von (ϵ,δ)-Differential Privacy lautet [34]:
Eine randomisierte Funktion M, welche einen Datensatz D auf einen Wertebereich R abbildet, erfüllt (ϵ,δ)-Differential Privacy, wenn für alle Datensätze D_1 und D_2, die sich in höchstens einem Datenpunkt unterscheiden, gilt:

$$Pr[M(D_1) \in R] \leq e^{\epsilon} \times Pr[M(D_2) \in R] + \delta \tag{3.2}$$

Konkret sagen die Definitionen aus, dass eine randomisierte Funktion, auf beiden Datensätzen angewendet, nahezu die gleichen Ergebnisse liefert. Dabei bestimmten ϵ und δ wie stark sich die Ergebnisse unterscheiden dürfen.

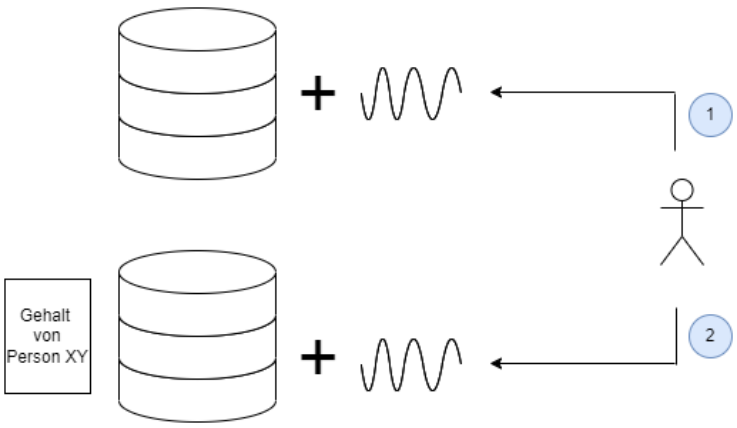

Abbildung 3.2: Beispiel Differential Privacy

Abbildung 3.2 zeigt exemplarisch, wie Differential Privacy anhand einer Abfrage des Durchschnittsgehalts eines Unternehmens aussehen könnte. Dabei führt ein Angreifer 2 Anfragen aus, auf die gleiche Datenbank, welche aber bei Anfrage 2 um einen zusätzlichen Gehaltseintrag eines neuen Mitarbeiters erweitert wurde. Würde kein Differential Privacy genutzt werden, könnte anhand der Anzahl der Mitarbeiter und der beiden durchschnittlichen Gehältern das exakte Gehalt des neuen Mitarbeiters berechnet werden. Wird nun Differential Privacy genutzt, wird ein zufälliges Rauschen über das Ergebnis der Anfrage gelegt, wo-

durch Anfrage 1 und 2 ein nahezu identisches Ergebnis ausliefern, abhängig der Parameter ϵ und δ.

Differential Privacy kann dabei an 3 unterschiedlichen Stellen der Machine Learning Pipeline genutzt werden:

- **Vorbereitung der Trainingsdaten:** Diese Methodik wird folgend in diesem Kapitel erläutert.
- **Trainingsalgorithmus:** Kapitel 3.4 beschreibt, welche Anpassung am Trainingsalgorithmus vorgenommen werden müssen, damit dieser Differential Privacy nutzen kann.
- **Vorhersage des Modells:** Wie der Output des Modells durch Nutzung von Differential Privacy geschützt werden kann, wird durch Kapitel 3.5.1 dargestellt.

3.2.3 Confussion

3.2.4 Synthetische Daten

3.3 Training des Modells

3.4 Training mit Differential Privacy

3.5 Betrieb des Modells

3.5.1 Differential Privacy der Vorhersagte

Kapitel 4

Konzeption PrivacyFlow

4.1 Vergleich zu anderen Frameworks/Lösungen

4.2 Wahl der Methoden

Kapitel 5

Implementierung PrivacyFlow

5.1 Architektur

5.2 Datensatz

5.3 Implementierung der Methoden

5.4 Evaluation + Vergleich mit Vanilla ML Pipeline - Performance & Technik

Kapitel 6

Evaluierung

6.1 Vergleich PrivacyFlow vs. normaler ML-Workflow

6.2 Schutz der Vertraulichkeit

Kapitel 7

Diskussion

7.1 Zusammenfassung der Ergebnisse

7.2 Handlungsempfehlung anhand der Ergebnisse

7.3 Offene Probleme + Ausblick auf zukünftige Forschung

Abbildungsverzeichnis

Tabellenverzeichnis

Auflistungen

Literatur

[1] L. Ouyang, J. Wu, X. Jiang u. a., *Training language models to follow instructions with human feedback*, 2022. arXiv: 2203.02155 [cs.CL].

[2] FirstMark, *The 2023 MAD (ML/AI/Data) Landscape*, https://mad.firstmark.com/, [Online; Abgerufen 09.Mai 2023], 2023.

[3] C. Cadwalladr und E. Graham-Harrison, *Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach*, https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election, [Online; Abgerufen 08.Mai 2023], März 2018.

[4] Bundesamt für Sicherheit in der Informationstechnik, „IT-Grundschutz-Kompendium," in Bundesamt für Sicherheit in der Informationstechnik, Bonn, 2023, S. 42, ISBN: 978-3-8462-0906-6.

[5] J. Corbet, *Class action against GitHub Copilot*, https://lwn.net/Articles/914150/, [Online; Abgerufen 09.Mai 2023], Nov. 2022.

[6] J. Bennett, S. Lanning u. a., „The netflix prize," in *Proceedings of KDD cup and workshop*, New York, Bd. 2007, 2007, S. 35.

[7] A. Narayanan und V. Shmatikov, „Robust De-anonymization of Large Sparse Datasets," in *2008 IEEE Symposium on Security and Privacy (sp 2008)*, 2008, S. 111–125. DOI: 10.1109/SP.2008.33.

[8] G. Ateniese, L. V. Mancini, A. Spognardi, A. Villani, D. Vitali und G. Felici, „Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers," *International Journal of Security and Networks*, Jg. 10, Nr. 3, S. 137–150, 2015. DOI: 10.1504/IJSN.2015.071829.

[9] K. Ganju, Q. Wang, W. Yang, C. A. Gunter und N. Borisov, „Property Inference Attacks on Fully Connected Neural Networks Using Permutation Invariant Representations," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, Ser. CCS '18, Toronto, Canada: Association for Computing Machinery, 2018, S. 619–633, ISBN: 9781450356930. DOI: 10.1145/3243734.3243834.

[10] D. Gopinath, H. Converse, C. Pasareanu und A. Taly, „Property Inference for Deep Neural Networks," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, S. 797–809. DOI: 10.1109/ASE.2019.00079.

[11] M. Fredrikson, S. Jha und T. Ristenpart, „Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, Ser. CCS '15, Denver, Colorado, USA: Association for Computing Machinery, 2015, S. 1322–1333, ISBN: 9781450338325. DOI: 10.1145/2810103.2813677.

[12] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li und D. Song, „The Secret Revealer: Generative Model-Inversion Attacks Against Deep Neural Networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Juni 2020.

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza u. a., „Generative adversarial networks," *Communications of the ACM*, Jg. 63, Nr. 11, S. 139–144, 2020.

[14] Z. He, T. Zhang und R. B. Lee, „Model Inversion Attacks against Collaborative Inference," in *Proceedings of the 35th Annual Computer Security Applications Conference*, Ser. ACSAC '19, San Juan, Puerto Rico, USA: Association for Computing Machinery, 2019, S. 148–162, ISBN: 9781450376280. DOI: 10.1145/3359789.3359824.

[15] R. Shokri, M. Stronati und V. Shmatikov, „Membership Inference Attacks against Machine Learning Models," *CoRR*, Jg. abs/1610.05820, 2016. arXiv: 1610.05820.

[16] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis und F. Tramèr, „Membership Inference Attacks From First Principles," *CoRR*, Jg. abs/2112.03570, 2021. arXiv: 2112.03570.

[17] C. A. Choquette-Choo, F. Tramer, N. Carlini und N. Papernot, „Label-Only Membership Inference Attacks," in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila und T. Zhang, Hrsg., Ser. Proceedings of Machine Learning Research, Bd. 139, PMLR, Juni 2021, S. 1964–1974.

[18] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos und D. Song, *The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks*, 2019. arXiv: 1802.08232 [cs.LG].

[19] N. Carlini, F. Tramer, E. Wallace u. a., „Extracting Training Data from Large Language Models.," in *USENIX Security Symposium*, Bd. 6, 2021.

[20] B. Biggio, B. Nelson und P. Laskov, *Poisoning Attacks against Support Vector Machines*, 2013. arXiv: 1206.6389 [cs.LG].

[21] C. Yang, Q. Wu, H. Li und Y. Chen, *Generative Poisoning Attack Method Against Neural Networks*, 2017. arXiv: 1703.01340 [cs.CR].

[22] J. Guo und C. Liu, „Practical Poisoning Attacks on Neural Networks," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox und J.-M. Frahm, Hrsg., Cham: Springer International Publishing, 2020, S. 142–158, ISBN: 978-3-030-58583-9.

[23] F. Tramèr, R. Shokri, A. S. Joaquin u. a., *Truth Serum: Poisoning Machine Learning Models to Reveal Their Secrets*, 2022. arXiv: 2204.00032 [cs.CR].

[24] T. Li, A. K. Sahu, A. Talwalkar und V. Smith, „Federated learning: Challenges, methods, and future directions," *IEEE signal processing magazine*, Jg. 37, Nr. 3, S. 50–60, 2020.

[25] L. Melis, C. Song, E. De Cristofaro und V. Shmatikov, „Exploiting Unintended Feature Leakage in Collaborative Learning," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, S. 691–706. DOI: 10.1109/SP.2019.00029.

[26] L. Zhu, Z. Liu und S. Han, „Deep Leakage from Gradients," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox und R. Garnett, Hrsg., Bd. 32, Curran Associates, Inc., 2019.

[27] B. Zhao, K. R. Mopuri und H. Bilen, „iDLG: Improved Deep Leakage from Gradients," *CoRR*, Jg. abs/2001.02610, 2020. arXiv: 2001.02610.

[28] B. Hitaj, G. Ateniese und F. Pérez-Cruz, „Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning," *CoRR*, Jg. abs/1702.07464, 2017. arXiv: 1702.07464.

[29] L. Sweeney, „k-anonymity: A model for protecting privacy," *International journal of uncertainty, fuzziness and knowledge-based systems*, Jg. 10, Nr. 05, S. 557–570, 2002.

[30] W. C. Jessica Li, *Titanic - Machine Learning from Disaster*, 2012.

[31] A. Machanavajjhala, D. Kifer, J. Gehrke und M. Venkitasubramaniam, „L-Diversity: Privacy beyond k-Anonymity," *ACM Trans. Knowl. Discov. Data*, Jg. 1, Nr. 1, 3–es, März 2007, ISSN: 1556-4681. DOI: 10.1145/1217299.1217302.

[32] N. Li, T. Li und S. Venkatasubramanian, „t-Closeness: Privacy Beyond k-Anonymity and l-Diversity," in *2007 IEEE 23rd International Conference on Data Engineering*, 2007, S. 106–115. DOI: 10.1109/ICDE.2007.367856.

[33] C. Dwork, „Differential Privacy," in *Automata, Languages and Programming*, M. Bugliesi, B. Preneel, V. Sassone und I. Wegener, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, S. 1–12, ISBN: 978-3-540-35908-1.

[34] C. Dwork und A. Roth, „The Algorithmic Foundations of Differential Privacy," *Found. Trends Theor. Comput. Sci.*, Jg. 9, Nr. 3–4, S. 211–407, Aug. 2014, ISSN: 1551-305X. DOI: 10.1561/0400000042.

Glossar

library A suite of reusable code inside of a programming language for software development. i

shell Terminal of a Linux/Unix system for entering commands. i