Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Visueller Vergleich von Klassifizierungen von verschiedenen Machine-Learning-Modellen

Komail Mohammadi

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Thomas Ertl
Betreuer/in:	Franziska Becker, M.Sc., Dr. Tanja Blascheck, Dipl.-Inf. Christoph Müller
Beginn am:	11. März 2020
Beendet am:	06. November 2020

Kurzfassung

Über einen Command & Control-Channel kommunizieren Bots mit ihrem Botmaster, der ihnen über diesen Channel Befehle sendet. Um das Blockieren dieser Channels zu erschweren, werden Domain-Generation Algorithms (DGAs) verwendet. Diese Algorithmen erzeugen periodisch Domänennamen, über die ein neuer Channel aufgebaut wird, falls der Alte blockiert wurde. Zur automatisierten Erkennung und Klassifizierung solcher Domänennamen sind Machine-Learning-Modelle entwickelt worden. Die Verbesserung dieser Modelle erfordert den Vergleich ihrer Ergebnisse. Ein Hindernis hierbei ist die große Anzahl von Klassen bei den nicht binären ML-Modelle. Um den Vergleich zu vereinfachen, werden die Klassen anhand ihrer DGAs geclustert und die Ergebnisse durch Histogramme in Kombination mit Boxplots visualisiert. Das entwickelte Konzept ermöglicht die Analyse der Gesamt- und Klassenperformance sowie der Performance auf Intanzebene.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Algorithmen werden zum Lösen von Problemen, wie z.B. das Sortieren von Daten, verwendet. Jedoch gibt es Probleme, für die die Erstellung eines Algorithmus schwierig ist. Ein Beispiel hierfür ist die Erkennung von Spam-Mails. Die Ausgabe des Algorithmus ist bekannt, aber die Transformierung der Eingabe zur Ausgabe nicht. In solchen Fällen ist die Verwendung von Machine Learning (ML) von Vorteil. Beim ML lernt der Computer, durch die Verwendung von Trainingsdaten, das Problem selbst zu lösen. Für die Erkennung von Spam-Mails würde eine große Menge von Mails, die sowohl Spam-Mails als auch normale Mails enthält, für das Training verwendet werden. Auf Basis des Trainings ist dann die Vorhersage für neue Mails möglich. [Alp14]
Ein weiteres Anwendungsgebiet von ML ist das Erkennen von Algorithmically Generated Domains (AGDs). Das sind Domänennamen, die von Domain-Generation Algorithms (DGAs) periodisch erstellt werden, um das Blockieren von Command & Control-Channels (C&C-Channels) zu verhindern. In Botnets werden diese C&C-Channels zur Kommunikation mit dem Botmaster verwendet. Der Botmaster sendet über diese Channels Befehle an die infizierten Computer, um sie für seine Zwecke einzusetzen. [FCT11]
Um die ML-Modelle zu verbessern, die AGDs erkennen und klassifizieren, ist der Vergleich ihrer Ergebnisse untereinander notwendig. Zur Analyse der Performance von ML-Modellen existieren bereits verschiedene Möglichkeiten. Ein Beispiel hierfür sind Performance-Metriken wie Accuracy oder Precision [FHM09]. Das Problem solcher Metriken ist aber, dass Erkenntnisse über das Verhalten des Modells nicht erkennbar sind. Daher wurden zur genaueren Analyse von ML-Modellen Visualisierungs-Tools [AHH+14][RAL+17][GBYH20] entwickelt. Ein Problem mit einigen dieser Tools ist, dass sie nicht für große Anzahlen von Klassen geeignet sind. Die Multiclass-Modelle zur Klassifizierung von Domänennamen verwenden nämlich 92 Klassen. Weitere Probleme sind, dass einige Tools keine Einsichten in das Verhalten des Modells geben oder die Analyse aufwändig ist, da die Visualisierungen zuerst an das Modell angepasst werden müssen.
Das in dieser Arbeit entwickelte Konzept ermöglicht den Vergleich von ML-Modellen mit vielen Klassen. Außerdem ist die Analyse der Gesamt- und Klassenperformance sowie der Performance auf Instanzebene durchführbar. Um die Anzahl der zu vergleichenden Klassen zu verringern, werden sie anhand ihrer DGAs geclustert und durch eine passende Visualisierung dargestellt. Diese Visualisierungen bieten Interaktionen, mit denen die Ergebnisse der Clusterings untersucht werden können.

2 Grundlagen

2.1 Machine Learning

Bei der Mustererkennung (Pattern Recognition) werden automatisch Regelmäßigkeiten in Daten gesucht. Die Suche erfolgt durch die Verwendung von Computer-Algorithmen. Die dadurch gefundenen Regelmäßigkeiten können dann zur Erfüllung verschiedener Aufgaben herangezogen werden können. Eine mögliche Aufgabe hierzu ist das Einteilen von Daten in verschiedene Kategorien, das am Beispiel der Erkennung von handgeschriebenen Ziffern, wie in Abbildung 2.1 zu sehen, demonstriert wird. Dargestellt ist jede Ziffer durch ein Bild mit 28×28 Pixeln, woraus ein Vektor $\mathbf{x}$ mit 784 Zahlen gebildet werden kann. Dieser Vektor dient als Eingabe für eine Maschine. Die Maschine soll auf diese Eingabe die Ziffer ausgeben, die auf dem Bild zu sehen ist. Basierend auf den Formen der Striche können feste Regeln oder Heuristiken abgeleitet werden, mit denen das Problem gelöst werden soll. Jedoch existieren viele verschiedene Handschriften, was zu einer Vielzahl von Regeln und Ausnahmen führt und damit zu schlechten Ergebnissen. [Bis06]
Um bessere Ergebnisse zu erhalten, kann Machine Learning (ML) verwendet werden und ist nach Alpaydin [Alp14] wie folgt definiert:

> "Machine learning is programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model may be predictive to make predictions in the future, or descriptive to gain knowledge from data, or both." (Alpaydin 2014: 3)

Für das Problem der Erkennung von handgeschriebenen Ziffern wird also ein großes Trainingsset mit verschiedenen handgeschriebenen Ziffern als Vektoren verwendet, womit die Parameter des Modells optimiert werden. Dazu muss vorher jedem Eingabevektor die korrekte Ziffer zugewiesen

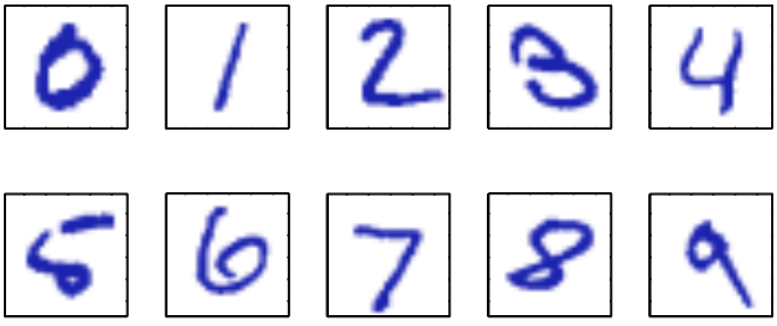

Abbildung 2.1: Beispiele für handgeschriebene Ziffern von 0 bis 9. [Bis06]

werden, woraus ein Vektor erstellt wird. Dieser heißt auch Target-Vector. Das trainierte Modell kann dann auf die Eingabe eines unbekannten Bildes die Ziffer, die darauf zu sehen ist, voraussagen. [Bis06]

2.1.1 Typen von Machine Learning

Machine Learning lässt sich in folgende Typen unterteilen [Mur12]:

Predictive/Supervised Learning Gegeben ist ein vorklassifiziertes Trainingsset $T = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ der Größe N. Gesucht ist eine Abbildung von den Eingaben $\mathbf{x}$ auf die Ausgaben y. Hierbei ist jede Eingabe $\mathbf{x}_i$ meist ein D-dimensionaler Vektor aus Zahlen und wird oft als Feature bezeichnet. Die Eingabe kann auch aus komplexeren Objekten, wie z.B. aus Bildern oder Sätzen, bestehen. Dies gilt auch für die Ausgaben, aber meistens ist y_i eine kategoriale oder nominale Variable aus einer endlichen Menge. Es wird von Klassifizierung oder Pattern Recognition bei kategorialen Ausgaben y_i gesprochen und von Regression, wenn y_i ein reeller Wert ist. [Mur12]

Descriptive/Unsupervised Learning Gegeben sind Eingaben der Form $T = \{(\mathbf{x}_i)\}_{i=1}^{N}$, wobei im Gegensatz zum Supervised Learning kein Ausgabevektor mit angegeben wird. Gesucht sind interessante Muster in den Daten, was auch als Knowledge Discovery bezeichnet wird. [Mur12]

Reinforcement Learning Reinforcement Learning arbeitet nach dem Trial-and-Error-Prinzip. Ein Agent trifft in einer Umgebung Entscheidungen und erhält für jede Aktion, je nach Ausgang, eine Belohnung oder eine Strafe. Das Ziel ist die Maximierung der Summe der Belohnungen nach mehreren Durchläufen. Die Sequenz von Aktionen, die diese Summe maximiert, wird auch als die beste Policy bezeichnet. [Alp14]

2.1.2 Klassifizierung

Eine häufige Anwendung von Supervised Learning ist die Klassifizierung. Gesucht ist dabei eine Abbildung von den Eingaben $\mathbf{x}$ auf die Ausgaben y. Wenn in zwei Klassen eingeteilt werden soll, wird von binären Klassifizierungen gesprochen. Erfolgt die Einteilung in mehr als zwei Klassen, ist von einer Mehrklassen-Klassifizierung (Multiclass-Classification) die Rede. Es kann auch sein, dass einer Eingabe mehrere Klassenlabel zugeordnet werden können. Dann ist von einer Multi-Label-Klassifizierung die Rede. Die trainierten Modelle werden verwendet, um Predictions (Vorhersagen) zu unbekannten Eingaben zu treffen, was als Generalization bezeichnet wird. [Mur12]
Ein Beispiel für die Klassifizierung ist in Abbildung 2.2 zu sehen. Die Eingaben sind verschieden gefärbte Formen, die in die zwei Klassen "yes" (Label 1) und "no" (Label 0) aufgeteilt sind. In der Mitte ist eine Matrix der Features zu sehen, die die Eingaben beschreiben und rechts davon ist der Vektor der Trainingslabels abgebildet. [Mur12]
Nun wird versucht, den drei unklassifizierten Formen ein Label zuzuordnen. Der blauen Sichel würde das Label 1 zugeordnet werden, da alle anderen blauen Formen auch dieses Label tragen. Beim gelben Kreis ist die Zuordnung nicht eindeutig, da beide Klassen gelbe Formen und anders gefärbte Kreise enthalten. Der blaue Pfeil ist auch nicht eindeutig zuzuordnen, da die Klasse "no" keine blauen Formen und die Klasse "yes" keine Pfeile enthält. Um bei solchen Fällen trotzdem

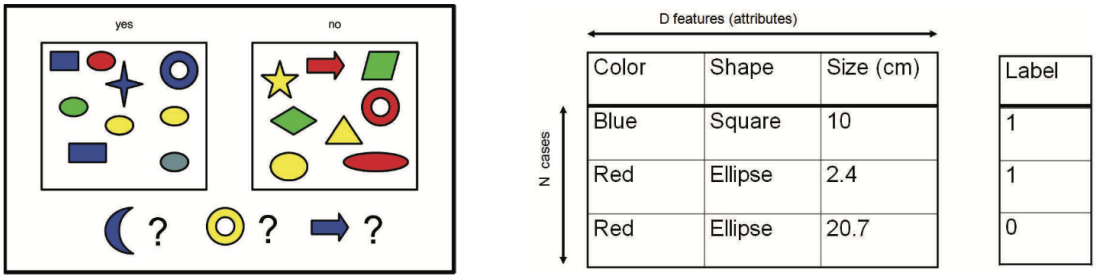

Abbildung 2.2: Beispiel einer Klassifizierung. Links: Vorklassifizierte Beispiele gefärbter Formen für das Training sowie 3 Testformen. Mitte: Trainingsdaten als $N \times D$ Matrix (Reihe i entspricht Featurevektor x_i). Rechts: Die entsprechenden Labels der Eingabedaten ($y_i \in \{0, 1\}$). [Mur12]

eine Entscheidung treffen zu können, ist es notwendig, eine Wahrscheinlichkeit für jede Klasse zurückzugeben. Die Klasse mit der höchsten Wahrscheinlichkeit ist dann die Prediction für diese Instanz. [Mur12]

2.1.3 Performance-Metriken

Um die Performance eines Classifiers/Modells auszuwerten, gibt es verschiedene Metriken. Im Folgenden werden einige Metriken näher beschrieben.

Accuracy

Die Accuracy gibt den Anteil von korrekten Predictions an, die der Classifier getroffen hat und wird mit der Formel

$$Accuracy = \frac{Anzahl\ korrekter\ Predictions}{Gesamtzahl\ von\ Predictions}$$

berechnet. [FHM09]

Precision

Die Precision gibt an, welcher Anteil der positiven Predictions korrekt ist und wird durch

$$Precision = \frac{Anzahl\ korrekter\ und\ positiver\ Predictions\ der\ Klasse}{Gesamtzahl\ positiver\ Predictions\ der\ Klasse}$$

berechnet. [FHM09]

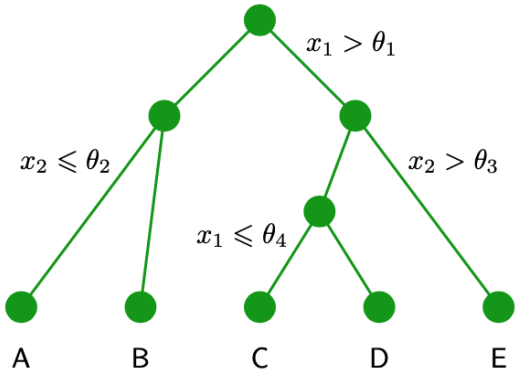

Abbildung 2.3: Beispiel eines Binärbaums, der den Eingaberaum in 5 Regionen aufteilt. [Bis06]

Recall

Der Recall berechnet sich durch die Formel

$$Recall = \frac{Anzahl\ korrekter\ und\ positiver\ Predictions\ der\ Klasse}{Gesamtzahl\ positiver\ Instanzen\ der\ Klasse}$$

und gibt an, welcher Anteil der Instanzen, die wirklich zu einer Klasse gehören, auch dieser zugewiesen wurden. [FHM09]

F-measure

Das F-measure lässt sich aus der Precision und dem Recall mit der Formel

$$F - measure = \frac{2 * Recall * Precision}{Recall + Precision}$$

berechnen. [FHM09]

2.1.4 Classification and regression trees (CART)

CARTs bestehen aus Binärbäumen. An jedem Knoten wird der Eingaberaum in zwei Regionen aufgeteilt. In Abbildung 2.3 ist dieser Prozess dargestellt. Dort ist ein Baum abgebildet, der den Eingaberaum in 5 Regionen aufteilt. Dazu erfolgt an jedem Knoten der Vergleich eines Wertes des Eingabevektors mit einer Schranke. Je nach Ausgang des Vergleichs wird am nächsten Knoten der Vorgang wiederholt, bis eine Region erreicht ist. [Bis06] Jede Region kann nun mit einer entsprechenden Antwort, wie z.B. einer Wahrscheinlichkeitsverteilung oder einem Klassenlabel, versehen werden [Mur12].

Random Forest

Ein Nachteil von CARTs ist, dass Bäume instabil sind. Kleine Änderungen in den Eingabedaten wirken sich auf die gesamte Baumstruktur aus. Das liegt daran, dass Fehler im oberen Teil des Baumes den Rest des Baumes beeinflussen. Um dem entgegenzuwirken, kann der Durchschnittswert aus mehreren Predictions für eine Instanz berechnet werden. Diese Technik wird Bagging (für "bootstrap aggregating") genannt. Dabei werden mehrere Bäume auf verschiedene Teilmengen der Daten trainiert, die zufällig ausgewählt werden und dabei Wiederholungen erlaubt sind. Diese Methode kann jedoch zu stark zusammenhängenden Predictions verschiedener Bäume führen. Um diese Zusammenhänge zu reduzieren, wird nicht nur eine zufällig generierte Teilmenge aus den Eingabedaten verwendet, sondern auch eine zufällig gewählte Teilmenge der Features. Dieses Verfahren ist auch als Random Forests bekannt. [Mur12]

Support Vector Machine (SVM)

Die Repräsentation jeder Instanz erfolgt durch einen Vektor im Featureraum. Um nun die Instanzen in zwei Klassen aufzuteilen, wird eine Hyperebene im Featureraum platziert, der die Instanzen voneinander trennt. Bei der Platzierung der Hyperebene wird das Large-Margin-Prinzip verwendet. Der Margin ist der senkrechte Abstand zwischen der Hyperebene und dem nächstgelegenen Vektor. Die Hyperebene wird so platziert, dass der Margin maximal ist. In Abbildung 2.4 ist dieses Prinzip dargestellt. [Bis06]
Für die Platzierung der Hyperebene werden nicht alle Vektoren benötigt, sondern nur die, die zu der Ebene am nächsten liegen. Diese Vektoren heißen auch Support Vectors (Stützvektoren). Die Vektoren dahinter beeinflussen nicht das Ergebnis und werden daher nicht für die Berechnungen benötigt. [Bis06]
Die Vektoren müssen nicht unbedingt im Featureraum linear trennbar sein. Die Platzierung einer Hyperebene ist dann nicht möglich. Um den nicht linearen Fall abzudecken, wird der Kernel-Trick verwendet. Dabei werden die Vektoren mithilfe einer nicht linearen Transformation in einen neuen Raum überführt. Im neuen Raum wird dann versucht die Hyperebene zu platzieren. Die lineare Ebene im neuen Raum ergibt eine nicht lineare Ebene im ursprünglichen Raum. Der Kernel-Trick beinhaltet auch den Einsatz von Kernelfunktionen. Sie ermöglichen die Transformation in einen neuen Raum, ohne die Transformation für die Instanzen ausrechnen zu müssen. Dies verringert die Rechenlast. [Alp14]

2.1.5 Neural Network

Ein Neural Network ist eine Folge von funktionalen Transformationen. Aus den Eingabevariablen $x_1, \ldots, x_D$ werden zunächst M Linearkombinationen

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

mit $j = 1, \ldots, M$ berechnet, die auch Activations genannt werden. Der Exponent gibt den Layer im Netzwerk an. Folglich steht (1) für den ersten Layer. $w_{ji}^{(1)}$ sind die Gewichte und $w_{j0}^{(1)}$ die Bias. Jede Activation wird dann durch eine Activation Function h umgewandelt, deren Ausgaben

$$z_j = h(a_j)$$

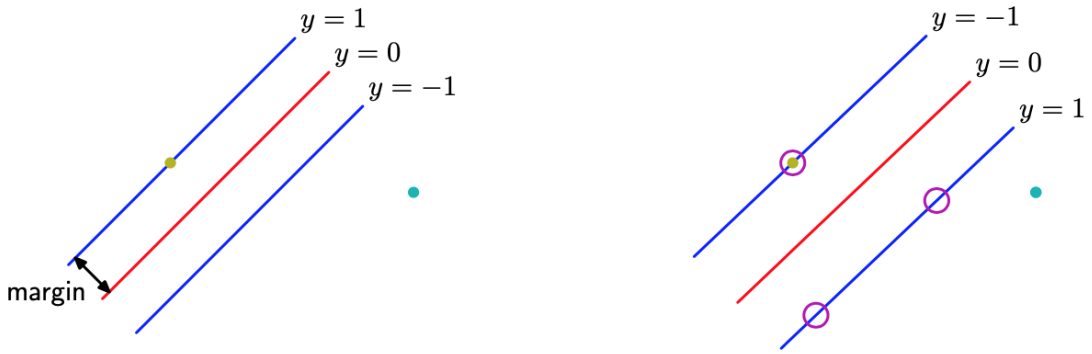

Abbildung 2.4: Links: Darstellung des Margins. Der Margin ist der senkrechte Abstand zwischen der Hyperebene und dem nächstgelegenen Vektor. Rechts: Platzierung der Hyperebene, sodass der Margin maximal ist. Die Kreise zeigen die Support Vectors (Stützvektoren) an, die für die Platzierung der Ebene benötigt werden. [Bis06]

Hidden Units heißen. Diese Ausgaben werden zu Output Unit Activations

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

mit $k = 1, \ldots, K$ kombiniert, wobei K die Gesamtzahl der Ausgaben angibt. Um schließlich die Network Outputs y_k zu erhalten, werden die Output Unit Activations mit einer passenden Activation Function transformiert. Der Gesamtvorgang wird auch als Forward Propagation bezeichnet, da sich die Outputs nur in eine Richtung bewegen. Das Modell des Two-Layer Neural Networks ist in Abbildung 2.5 zusammengefasst als Diagramm dargestellt. [Bis06]
Sind auch rückwärtige Verbindungen zu den Knoten und zu anderen Layern erlaubt, ist von einem **Recurrent Neural Network** (RNN) die Rede. [Mur12]

Convolutional Neural Network (CNN)

Es muss nicht jedes Input-Feature einzeln informativ sein, wie z.B. nicht jeder Pixel in einem Bild sehr informationsreich ist, sondern die Kombination der Pixel. Hier kann die Verwendung von Convolutional Neural Networks von Vorteil sein. Bei solchen Netzwerken haben die Hidden Units lokale rezeptive Felder, die über das Bild hinweg die Gewichte bündeln oder teilen. Dadurch wird die Parameteranzahl reduziert und nützliche Features, die in einem Bildbereich entdeckt wurden, können in anderen Layern wiederverwendet werden und müssen nicht immer wieder neu erlernt werden. Diese Eigenschaft wird auch Translation Invariance genannt. [Mur12]

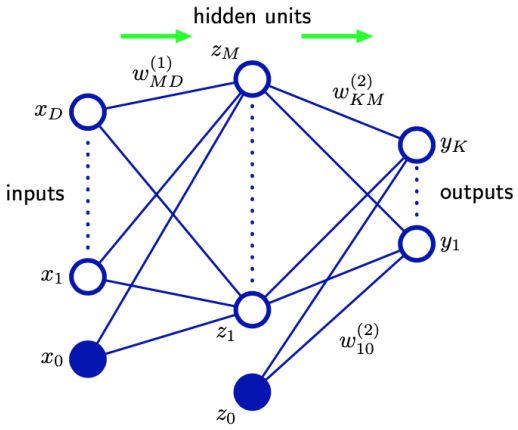

Abbildung 2.5: Darstellung eines Two-Layer Neural Networks. Die grünen Pfeile zeigen den Informationsfluss im Netzwerk an. Die Variablen sind als Knoten dargestellt und die Gewichte durch Verbindungen zwischen den Knoten. Die Bias sind durch Verbindungen von x_0 und z_0 aus kommend erkennbar. [Bis06]

2.2 Clustering

Clustering ist ein anderer Begriff für Unsupervised Classification. Beim Clustering wird versucht eine Menge von Datenobjekten in Cluster aufzuteilen, wobei ein Cluster für eine Gruppe, Untermenge oder Kategorie steht. Die Objekte innerhalb eines Clusters sollten möglichst ähnlich zueinander sein und Objekte aus unterschiedlichen Clustern möglichst unähnlich. [XW09]

2.2.1 Proximitätsmaße für kontinuierliche Variablen

Eine bekannte und häufig genutzte Metrik zum Bestimmen der Distanz ist die **Euklidische Distanz**

$$D(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{l=1}^{d} |x_{il} - x_{jl}|^{1/2} \right)^2 ,$$

die ein Spezialfall der **Minkowski-Distanz**

$$D(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{l=1}^{d} |x_{il} - x_{jl}|^{1/p} \right)^p$$

ist. Die Verwendung der euklidischen Distanz beim Clustering führt zu hypersphärischen Clustern und macht die Cluster für Translationen und Rotationen invariant. Bei der Verwendung unterschiedlicher Einheiten für die Features können Features mit großen Werten und Varianzen andere Features dominieren. Außerdem können Transformationen dazu führen, dass sich die Abstandsbeziehungen zwischen den Instanzen verändern. Durch die Normalisierung der Daten kann dies jedoch verhindert werden. [XW09]
Zwei andere Spezialfälle der Minkowski-Distanz sind für $p = 1$ die **Manhattan-Distanz**

$$D(\mathbf{x}_i, \mathbf{x}_j) = \sum_{l=1}^{d} |x_{il} - x_{jl}|$$

und für $p \to \infty$ die **Sup-Distanz**, die mit der Formel

$$D(\mathbf{x}_i, \mathbf{x}_j) = \max_{1 \le l \le d} |x_{il} - x_{jl}|$$

angegeben wird. [XW09]

2.2.2 Partitional Clustering

Beim Partitional Clustering ist die Clusteranzahl vorgegeben [XW09]. Die Instanzen werden vom Algorithmus in Partitionen aufgeteilt. Nach Beendigung des Algorithmus stellt eine Partition einen Cluster dar. Jeder Cluster enthält dann mindestens eine Instanz und jede Instanz ist eindeutig zugeordnet. [FAT+14]

K-Means Algorithmus

Der K-Means Algorithmus gehört zum Partitional Clustering und läuft mit der Vorgabe der Clusteranzahl K folgendermaßen ab [XW09]:

1. Wähle zufällig K Anfangszentren.
2. Berechne für jede Instanz die Distanz zu den Clusterzentren und weise sie dem Cluster zu, dessen Zentrum am nächsten ist.
3. Aktualisiere die Clusterzentren.
4. Die letzten zwei Schritte werden solange wiederholt, bis sich die Clusterzentren nicht mehr ändern.

Es kann entweder als Vorteil oder Nachteil gesehen werden, dass die Clusteranzahl vorzugeben ist. Dies hängt davon ab, welches Ziel beim Clustering verfolgt wird. K-Means funktioniert bei kompakten und hypersphärischen Clustern gut, hat aber bei anderen Clusterformen Probleme. Dafür ist K-Means effizient und auch für große Datensätze geeignet. [XW09]

2.3 Domain-Generation Algorithms (DGAs)

Nach Sood und Zeadally [SZ16] werden DGAs verwendet, um große Mengen von pseudozufälligen Domänennamen zu erzeugen, wobei ein vorberechneter Startwert benutzt wird, der dem Angreifer bekannt ist. Die so erzeugten Domänennamen heißen Algorithmically Generated Domains (AGDs). Da der Angreifer den Startwert kennt, kann er die selben AGDs erzeugen und somit eine der Domains registrieren, mit der sich die Malware verbinden und einen Kommunikationskanal erstellen kann, um so den Command & Control-Server (C&C-Server) für seine Zwecke einzusetzen. DGAs sind sozusagen ein Backup dafür, falls der Kommuniaktionskanal ausfällt. Der Angreifer kann auch einen Zeitplan festlegen, nach dem neue AGDs erstellt werden.
Es gibt verschiedene Strukturen, nach denen AGDs erzeugt werden. Ein Auflistung dieser Strukturen ist in Abbildung 2.6 zu finden, worauf 6 verschiedene Möglichkeiten zu sehen sind. Bei der ersten Struktur wird eine Dictionary verwendet, aus denen die Wörter für die Domain übernommen werden.

AGD structure	Examples
Dictionary of specific words	accelerateaccountant.in.net, accelerateactor.in.net, and accelerateactress.in.net
Use of dynamic DNS	agdcjbdaic.cnc.noip.com, bhgghbfhbe.cnc-thunder.noip.com, and bmpcvxlnydgmiotd.static.noip.co
Alphabetic layout	bkkfskayedynbcjnkxx.tw, kpugubkyukenqqtqljfgxwykc.tw, and hiimlliieoclwavt.nu
Numeric layout	0731131430.com, 0943562134.net, and 1457230987.biz
Alphanumeric layout	y3aaa48a7056d7075c3760cdbd90a75b8f.cc, z376dfe4955a257a78944864dd0158d172.ws, and c9cca04cec2588918820cf33ba4337cca8.hk
Hybrid layout	mdecub-ydyg.ru, mgefa-bugin.com, and 0372yaa.hycyeea.biz

Abbildung 2.6: Eine Auflistung verschiedener AGD-Strukturen. [SZ16]

Die zweite Struktur macht von dynamischen DNS Gebrauch. Das alphabetische und numerische Layout benutzt entweder nur Buchstaben oder nur Zahlen für die Domains. Das alphanumerische Layout dagegen verwendet sowohl Buchstaben als auch Zahlen. Das Hybrid-Layout ist eine Kombination aus den vorher genannten Strukturen. [SZ16]

2.4 Visualization

2.4.1 Information und Scientific Visualization

Sowohl bei der Information Visualization (InfoVis) als auch bei der Scientific Visualization (SciVis) werden mithilfe von Computern Visualisierungen aus den Daten erstellt, um neue Erkenntnisse aus ihnen zu gewinnen. Bei der SciVis sind die Ausgangsdaten sensorische Daten. Aus den Visualisierungen zu diesen Daten sollen Forscher Schlüsse zu ihren Fragestellungen ziehen können. Die Ausgangsdaten bei der InfoVis dagegen sind abstrakte textuelle und numerische Daten und Baum- oder Netzwerkstrukturen. Das Ziel ist es, die abstrakten Daten für Menschen verständlich zu machen, indem die visuellen Strukturen für das Auge begreifbar gemacht werden. [RJ13]

2.4.2 Visual Analytics

Das Ziel von Visual Analytics ist die einfache Untersuchung großer Datensätze. Um dies zu erreichen, verbindet Visual Analytics die automatisierte Analyse mit interaktiven Visualisierungen. Der Prozess von Visual Analytics ist in Abbildung 2.7 dargestellt. Im ersten Schritt des Prozesses erfolgt das Vorbereiten und Transformieren der Daten. Nach diesem Schritt kann entweder mit einer visuellen oder mit einer automatisierten Analyse fortgefahren werden. Bei der automatisierten Analyse kommen Data-Mining-Techniken zum Einsatz. Mit diesen Techniken werden Modelle aus den ursprünglichen Daten erstellt. Darauf folgt die Auswertung und Verfeinerung der Modelle durch die Interaktion mit den Daten. Die Visualisierungen ermöglichen die Evaluation der generierten Modelle. Am Ende des Prozesses erhofft sich der Benutzer einen Erkenntnisgewinn. [KKEM10]

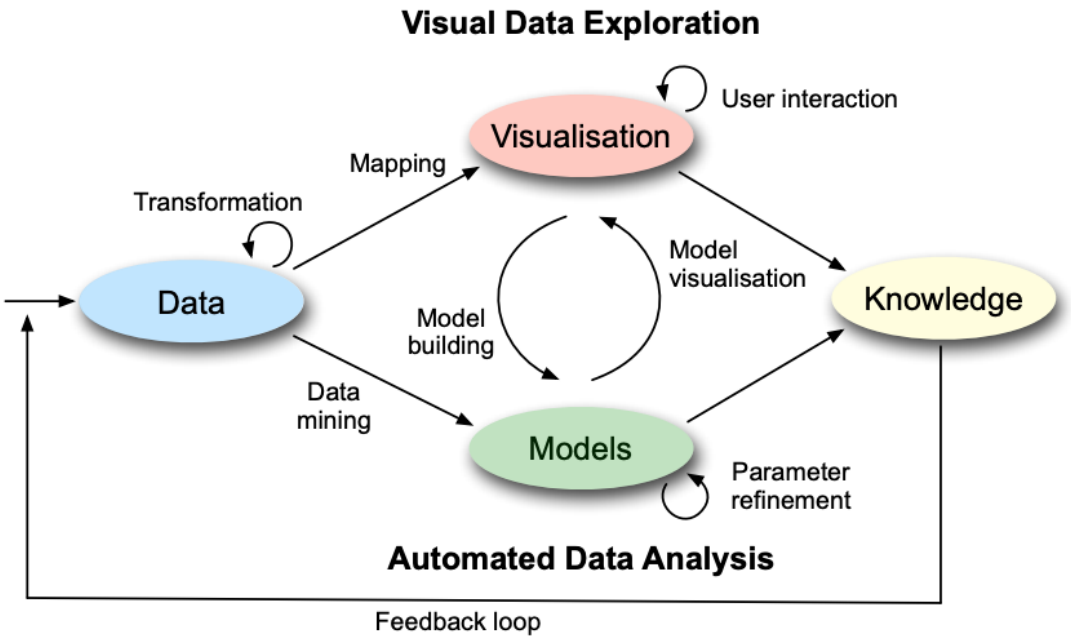

Abbildung 2.7: Der Prozess von Visual Analytics. Sind die Daten vorbereitet, kann zwischen visueller und automatisierte Analyse entschieden werden. Auch der Wechsel zwischen diesen Analysen ist möglich. Am Ende erfolgt der Erkenntnisgewinn aus den vorherigen Schritten. [KKEM10]

2.4.3 Boxplots

Boxplots geben die Verteilung eines Datensatzes wieder. Sie bestehen aus dem Maximum, dem Minimum, dem oberen und unteren Quartil sowie dem Median des Datensatzes. Die Quartile teilen den Datensatz in vier gleich große Teilmengen auf. Ein Boxplot wird meist wie in Abbildung 2.8 dargestellt. Die Box ist durch das untere und obere Quartil begrenzt, wobei der Median in dieser Box liegt. Sie enthält 50% der Daten. Die Linien (oder Whiskers) an den Enden der Box reichen bis zum Maximum und Minimum der Daten. Alternativ können die Linien auf das 1.5-fache des Interquartile-Ranges (IQR) begrenzt werden, um Ausreißer auszugrenzen. Der IQR ist der Bereich, in dem 50% der Daten liegen und wird durch die Differenz des oberen und unteren Quartils berechnet [Jas13]. Ausreißer in den Daten können durch entsprechende Symbole angezeigt werden. Andere Eigenschaften wie die Breite oder Farbe der Box hängen von den zu darstellenden Daten und der Verwendung des Boxplots ab. [Pot06]

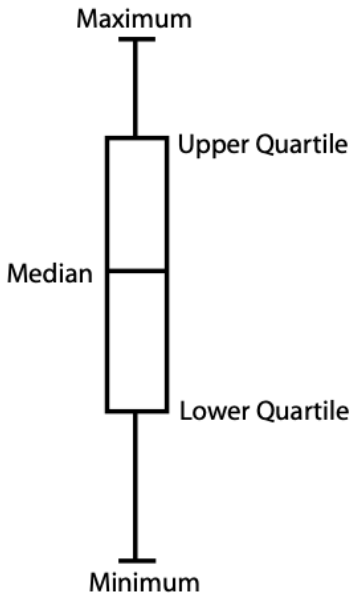

Abbildung 2.8: Darstellung eines Boxplots. [Pot06]

2.4.4 Parallel Coordinates

Zur Visualisierung von Daten mit vielen Dimensionen ist die Verwendung von Parallel Coordinates möglich. Die Darstellung der Daten, die als Vektoren gegeben sind, erfolgt als zweidimensionaler Graph. In diesem Graph wird für jede Dimension des Vektors eine vertikale Achse hinzugefügt. Die Vektoren werden als Polylines im Graph dargestellt, wobei die Punkte der Polyline auf den Achsen liegen. Ein Beispiel für Parallel Coordinates ist in Abbildung 2.9 zu sehen. [LZY17]

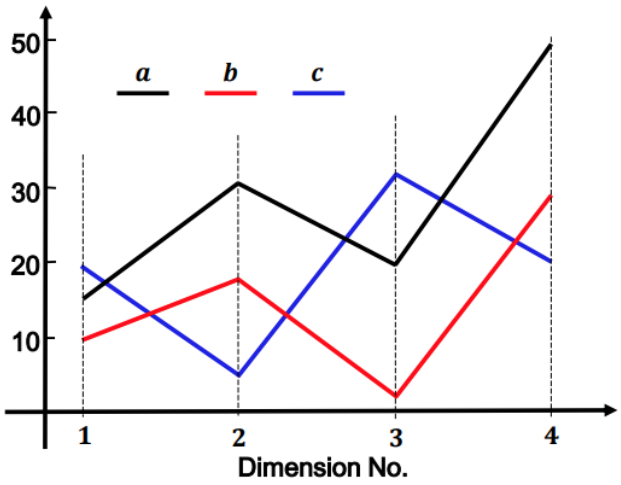

Abbildung 2.9: Beispiel für Parallel Coordinates. Die Punkte $a(15, 31, 20, 50)$, $b(10, 18, 2, 30)$ und $c(20, 5, 32, 20)$ sind als Polylines dargestellt. [LZY17]

3 Verwandte Arbeiten

In diesem Kapitel werden Arbeiten erwähnt, die sich bereits der Analyse von ML-Modellen angenommen haben. Dabei stehen Arbeiten im Fokus, die sich mit der Analyse von Multiclass-Classfiern beschäftigen.
ModelTracker [ACD+15] bietet die Möglichkeit, die Gesamtperformance eines Modells zu analysieren und dabei auch einzelne Instanzen zu untersuchen. Boxen werden zur Darstellung einzelner Instanzen verwendet, wobei grüne Boxen für korrekt klassifizierte Instanzen und rote Boxen für falsch klassifizierte Instanzen stehen. Diese sind an einer horizontalen Achse nach dem Wert ihres Prediction-Scores aufsteigend von links nach rechts angeordnet. Jedoch unterstützt ModelTracker nur die Analyse von binären Klassifizierungen. Für die Analyse der binären DGA-Classifier stellt dies kein Problem dar, aber für die Multiclass-Classifier ist ModelTracker ungeeignet.
Eine andere Herangehensweise bietet die Arbeit von Alsallakh et al. [AHH+14]. Das vorgestellte Analyse-Tool in dieser Arbeit erstellt verschiedene Visualisierungen für die Ergebnisse der Klassifizierung. Die erste Visualisierung ist ein Confusion-Wheel, das die Verteilung der Prediction-Scores der Instanzen einer Klasse als Histogramm darstellt. Dabei sind die Instanzen nach den Ergebnissen ihrer Klassifizierungen gefärbt. Eine weitere Visualisierung ermöglicht die Analyse der Features. Sie zeigt die Verteilung der Features für ausgewählte Instanzen an. Zusätzlich werden noch Histogramme und Scatterplots erzeugt. Sie dienen zur Veranschaulichung der Trennbarkeit von ausgewählten Instanzen nach ihren Features. Eine Darstellung des Analyse-Tools ist in Abbildung 3.1 zu sehen. Dieses Tool erlaubt den Vergleich von bis zu 20 Klassen in der Confusion-Wheel. Bei größeren Anzahlen von Klassen kann eine Teilmenge ausgewählt werden, die angezeigt werden soll. Die Multiclass-Classifier für DGAs arbeiten mit 92 Klassen. Das würde mehrere Vergleiche verschiedener Teilmengen erfordern. Das Ziel dieser Arbeit ist es u.a., die Gesamtperformance eines Modells in einer Visualisierung feststellen zu können und dabei keine Teilmengen auswählen zu müssen.
Eine mit dem eben genannten Tool verwandte Arbeit ist Squares [RAL+17]. Squares erstellt für jede Klasse eine Spalte und weist dieser eine Farbe zu. Zu jeder Spalte wird eine vertikale Achse mit dem Klassennamen am unteren Ende hinzugefügt. Die Achse gibt den Prediction-Score an, wobei der Score von 0 am unteren Ende und die 1 am oberen Ende ist. Für jede Instanz wird, wie bei ModelTracker, eine Box erstellt und nach dem Wert ihres Prediction-Scores an der Achse platziert. Auf der rechten Seite der Achse befinden sich die Instanzen, die dieser Klasse zugewiesen wurden. Richtige klassifizierte Instanzen erhalten eine voll ausgefüllte Box mit der Farbe ihrer Klasse. Falsch klassifizierte Instanzen werden gestreift dargestellt. Auf der linken Seite sind die Instanzen, die zu dieser Klasse gehören, aber einer anderen zugewiesen wurden. Sie werden mit der Farbe ihrer zugewiesenen Klasse umrandet. Neben der Darstellung als Boxen gibt es die Darstellung als Strips und Stacks, die die Boxen zusammenfassen. Sparklines von Parallel-Coordinates der Instanzen einer Klasse über den Achsen ermöglichen die Bestimmung von Between-Class-Confusions. Über die Boxen, Strips oder Stacks wird das Anzeigen der jeweiligen Instanzen als Tabelle unterstützt, sowie die Darstellung von Instanzen als Parallel-Coordinates, die über die Achsen gelegt werden. Das Ziel von Squares ist die Analyse von Classifiern mit 3-20 Klassen. Auch bei diesem Tool müssten

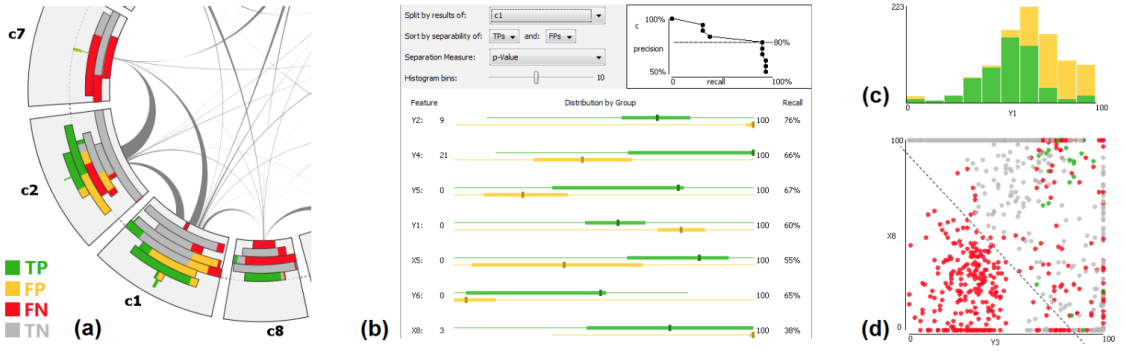

Abbildung 3.1: Das Analyse-Tool aus dem Paper von Alsallakh et al. [AHH+14]. (a) Die Confusion-Wheel, die die Verteilung der Predictions in den Klassen als Histogramme darstellt. (b) Möglichkeit zur Analyse der Features. (c, d) Histogramme sowie Scatterplots, die die Trennbarkeit von Instanzen nach Features erkennbar machen.

mehrere Teilmengen der Klassen zum Erfassen der Gesamtperformance verglichen werden. Squares stellt dennoch die Basis für diese Arbeit dar und wird zur Unterstützung der Multiclass-Classifier für DGAs angepasst.
Das What-If Tool [WPB+19] ermöglicht es, ML-Modelle mit geringem Programmieraufwand zu visualisieren und zu analysieren. Dabei bietet das Tool eine Breite von Visualisierungen an. Bespiele hierfür sind Confusion-Matrizen, Histogramme und Scatterplots, die auf Teilmengen des Datensatzes angewendet werden. Anwender können mit diesem Tool die Performance ihres Modells testen, die Bedeutung von Features untersuchen, das Verhalten eines Modells visualisieren und die Fairness des Modells messen. Dieses Tool erfordert aber Fachkenntnisse über ML und Data-Science. In dieser Arbeit soll die Analyse zielgerichteter als beim What-If Tool gestaltet werden, sodass der Nutzer einfach eine hilfreiche Visualisierung für die Analyse angezeigt bekommt, da die zu vergleichenden Modelle bekannt sind.
Das Ziel von Manifold[ZWM+19] ist die Interpretation, das Debuggen und der Vergleich von von ML-Modellen. Der Workflow von Manifold basiert auf den 3 Phasen Inspection, Explanation und Refinement. Diese Phasen sind oft in der Entwicklung und der Diagnose von Modellen anzutreffen. Die Visualisierung dafür basiert auf Zusammenfassungen, die aus Scatterplots bestehen, und geben einen Überblick über die Modellergebnisse. Zusätzlich wird eine tabellarische Ansicht erstellt, die die Analyse von Features ermöglicht. Dieses Tool ist in Abbildung 3.2 zu sehen. Manifold bietet keine Analyse der internen Vorgänge in den Modellen und betrachtet die zu vergleichenden Modelle als Blackbox. In dieser Arbeit soll auch die Analyse einzelner Klassen und Instanzen möglich sein, wobei die Features nicht im Fokus der Analyse stehen.
Ein weiteres Tool zum Vergleich von ML-Modellen ist Boxer [GBYH20]. Boxer wendet mehrere Classifier auf einen gemeinsamen Satz von Instanzen an. Das Ziel ist die Identifizierung von interessanten Teilmengen aus den Trainings- und Testinstanzen und der Vergleich von Classifiern anhand dieser Teilmengen. Dem Benutzer wird es ermöglicht, Views zusammenzustellen und zu koordinieren, wie in Abbildung 3.3 dargestellt. Dadurch können viele mögliche Szenarien bei der Entwicklung und Analyse von Classifiern abgedeckt werden. Zur Analyse müssen bei Boxer die Views passend kombiniert werden. Da die zu vergleichenden Modelle in dieser Arbeit die

Abbildung 3.2: Die Views von Manifold. (1) Der Vergleich verschiedener Modelle anhand von Scatterplots. (2, 3) Vergleich von Features für Teilmengen der Instanzen, die der Benutzer festgelegt hat. [ZWM+19]

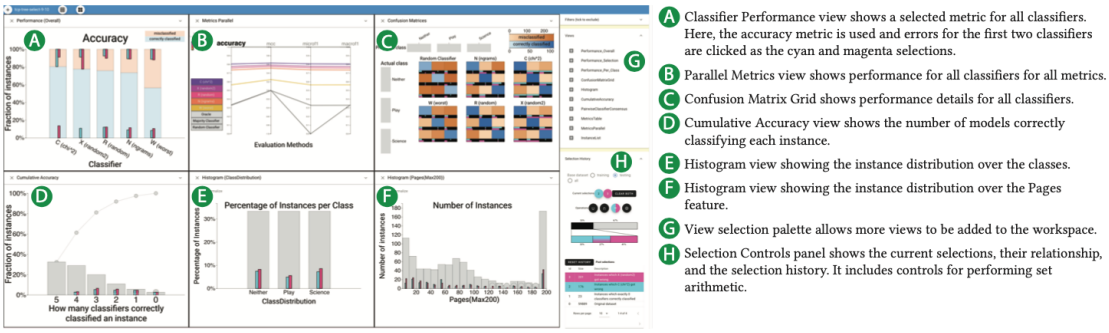

Abbildung 3.3: Darstellung von Boxer. Fünf Classifier werden verglichen, die zur Klassifizierung verschiedene Features verwenden. Dafür stehen acht unterschiedliche Views zur Verfügung. [GBYH20]

DGA-Classifier sind, kann entsprechend eine angepasste Visualisierung erstellt werden, wozu nicht das Kombinieren von Views notwendig ist. Dies soll den Analyseprozess für den Anwender vereinfachen.

4 Konzept

4.1 Ziel

Das Ziel dieser Arbeit ist die Entwicklung eines Analysesystems zum visuellen Vergleich der Klassifizierungen verschiedener ML-Modelle anhand von Testdatensätzen. Die Modelle, die zu vergleichen sind, wurden entwickelt um AGDs zu klassifizieren. Eine Herausforderung hierbei ist die große Anzahl an Klassen, denen die AGDs zugeordnet werden. Zusätzlich sollen im Analysesystem die Daten unter Einbeziehung von DGAs geclustert werden. Außerdem soll die visuelle Analyse der Ergebnisse des Clusterings durch passende Interaktionsmöglichkeiten möglich sein. Bei der Erstellung des Konzepts werden folgende Designziele angestrebt, die von Ren et al. [RAL+17] anhand einer Umfrage über das Vorgehen und die Schwierigkeiten beim Erstellen von Multiclass-Classifiern bestimmt wurden:

Z1: Performance-Analyse auf verschiedenen Ebenen Es soll die Analyse der Gesamtperformance und die Analyse auf Klassen- sowie auf Instanzebene ermöglicht werden, wobei die Analyse auf Instanzebene eine geringere Priorität hat als die ersten zwei Ebenen. Dies ermöglicht dem User, Schwerpunkte beim Verbessern der Performance zu setzen.

Z2: Unabhängigkeit von Metriken Wenn die Visualisierung unabhängig von verschiedenen Performance-Metriken ist, kann eine größere Bandbreite von Klassifizierungsproblemen abgedeckt werden, da vom Problem abhängig ist, welche Metriken zu optimieren sind.

Z3: Verbindung von Performance und Daten Von der Visualisierung der Performance aus soll der interaktive Zugriff auf die Daten ermöglicht werden, um den ständigen Wechsel zwischen Visualisierung und Daten zu verringern. Um dies zu realisieren, sollten Visualisierung und Daten auf demselben Screen angezeigt werden.

4.2 Squares: Performance-Visualisierung für Multiclass-Klassifizierungen

Um den visuellen Vergleich verschiedener ML-Modelle zu ermöglichen, sollten die Visualisierungen unabhängig von den Modellen sein, da die Verwendung unterschiedlicher Visualisierungen das Vergleichen erschwert. Eine Visualisierung, die den Vergleich unterschiedlicher Modelle unterstützt und auch mit Multiclass-Klassifizierungen zurecht kommt, ist Squares [RAL+17], worauf diese Arbeit basiert. Deshalb wird das Konzept von Squares im Folgenden erklärt.

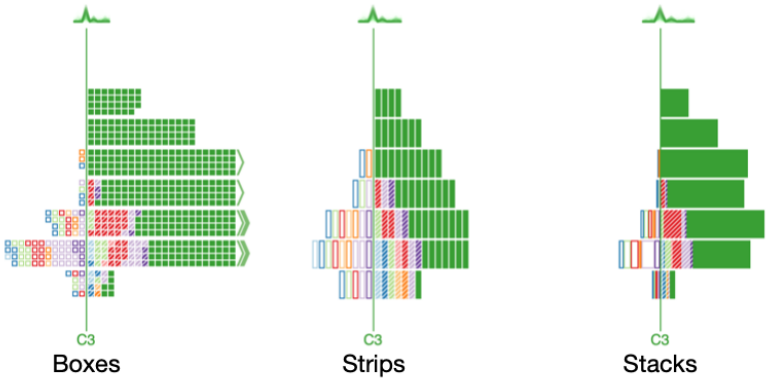

Abbildung 4.1: Darstellungsweisen von Klassen in Squares. Boxen repräsentieren einzelne Instanzen. Strips fassen eine festlegbare Zahl von Boxen zusammen und Stacks fassen alle Boxen zusammen. [RAL+17]

4.2.1 Darstellung der Klassen

Squares erstellt für jede Klasse eine Spalte und weist dieser eine Farbe zu. Zu jeder Spalte wird eine vertikale Achse, die den Prediction-Score-Bereich angibt, mit dem Klassennamen am unteren Ende hinzugefügt. Dabei ist der geringste Prediction-Score am unteren Ende der Achse und der Höchste am oberen Ende.
Für jede Instanz aus den Daten wird eine Box erstellt und zu ihrer vom Modell zugewiesenen Klasse auf der rechten Seite der Achse nach dem Wert ihres Prediction-Scores hinzugefügt. Jede Box erhält dabei die Farbe ihrer wahren Klasse, also der Klasse, zu der sie in Wirklichkeit gehört. Korrekte Klassifizierungen werden voll ausgefüllt und inkorrekte Klassifizierungen werden gestreift dargestellt. Auf der linken Seite der Achse befinden sich Boxen, die eigentlich zu dieser Klasse gehören, aber einer anderen zugewiesen wurden und werden nicht mit einer Farbe ausgefüllt. Stattdessen erhalten sie eine Umrandung mit der Farbe ihrer zugewiesenen Klasse. In Abbildung 4.1 ist links ein Beispiel hierfür zu sehen.
Neben der Darstellung mit den Boxen gibt es noch die Ansicht mit Strips und Stacks für größere Datensätze. Jeder Strip fasst eine festlegbare Zahl von Boxen zusammen. Ein Stack dagegen fast alle Boxen zusammen und gibt die Verteilung der Instanzen wieder. Bei den Boxen werden Pfeile am Ende eines Balkens hinzugefügt, um anzuzeigen, dass weitere Boxen existieren. Beispiele für diese Visualisierungen sind in Abbildung 4.1 mittig und rechts zu sehen.

4.2.2 Visualisierungen auf Instanzebene und Between-Class-Confusion

Wenn das verwendete ML-Modell Prediction-Scores für alle Klassen ausgibt, werden diese mit einer Polyline angezeigt, wenn der User die Maus über eine Box bewegt oder auf die Box klickt. Die eingeblendete Polyline schneidet die Achse für jede Klasse auf Höhe des Scores der Instanz für diese Klasse. Mehrere Instanzen werden mittels Parallel-Coordinates dargestellt, wie in Abbildung 4.2 dargestellt.
Zur Bestimmung der Between-Class-Confusion, wobei es um die Klassen geht, die oft miteinander verwechselt werden, erfolgt die Anzeige von Sparklines für jede Klasse über ihre Achse. Die

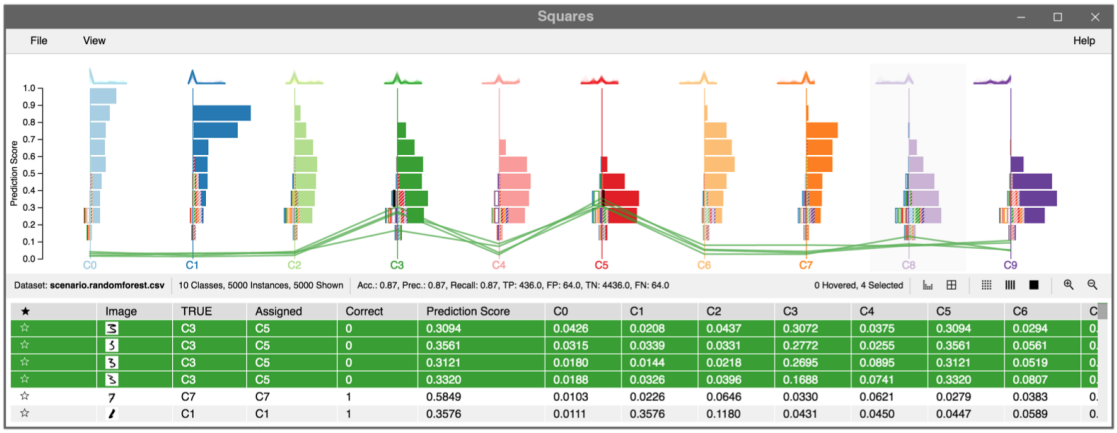

Abbildung 4.2: Beispiel einer Squares-Visualisierung. Für die ausgewählten Instanzen werden die entsprechenden Polylines angezeigt. Durch das Klicken auf Boxen, Stripes oder Stacks können in der Tabelle die dazugehörigen Instanzen eingeblendet werden. Dies ist möglich, da die Verbindung von Daten und Visualisierung in beide Richtungen gegeben ist. [RAL+17]

Sparkline zeigt die Parallel-Coordinates von allen Instanzen an, die dieser Klasse vom Modell zugewiesen wurden. Hohe Spitzen bei mehreren Klassen weisen auf Class-Confusions hin. In Abbildung 4.2 sind die beschriebenen Sparklines zu sehen.

4.2.3 Verbindung von Performance und Daten

Durch das Klicken auf Boxen, Stripes oder Stacks werden die jeweiligen Instanzen in einer Tabelle unter der Squares-Visualisierung angezeigt. Die Verbindung von Visualisierung und Daten ist dabei in beide Richtungen gegeben, sodass von der Tabelle aus Instanzen ausgewählt und in der oberen Visualisierung angezeigt werden können, wie in Abbildung 4.2 dargestellt. Zusätzlich können die Daten gefiltert werden, um interessante Teilmengen herauszufinden.

4.3 Probleme und Lösungen

Squares ist auf die Analyse von ML-Modellen mit 3-20 Klassen ausgelegt. Die Modelle für die Klassifizierungen von DGAs arbeiten aber mit 92 Klassen. Einer dieser Klassen ist die Klasse "benign" und enthält die Domänennamen, die nicht von einem DGA erzeugt wurden. Deshalb ist Squares in dieser Form für diese Modelle nicht anwendbar. Um einen Vergleich trotzdem zu ermöglichen, wird Squares für diesen Zweck angepasst. Im Folgenden werden die auftretenden Probleme sowie Lösungen näher erläutert.

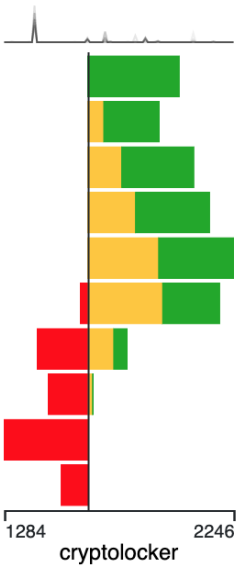

Abbildung 4.3: Darstellung der Klasse "cryptolocker" mit Stacks. Rot steht für FN, Gelb für FP und Grün für TP. Die Balken geben die Anzahl der Instanzen im jeweiligen Prediction-Score-Bereich an. Auf der X-Achse ist für jede Seite die maximale Anzahl an Instanzen aller Balken angegeben. Über der Klasse ist eine Sparkline zu sehen, die die Parallel Coordinates darstellen.

4.3.1 Darstellung der Klassen

Das erste Problem ist die Darstellung der Klassen. Squares weist jeder Klasse eine individuelle Farbe zu. Für die DGA-Modelle würden also 92 verschiedene Farben benötigt werden oder die Farben müssten sich wiederholen. Aber anhand der Farbe auf eine Klasse zu schließen wäre dann sehr schwierig, weshalb die Farben weggelassen werden. Außerdem wird nur die Darstellung als Stacks verwendet, da die verwendeten Datensätze (über 500000 Instanzen) sehr groß sind.
Eingeführt werden die Farben Rot, Gelb und Grün für die Stacks, in Anlehnung an die Arbeit von Alsallakh et al. [AHH+14]. Rot steht für die Instanzen, die False Negative (FN) sind, also zu dieser Klasse gehören, aber einer anderen zugewiesen wurden. Diese Instanzen befinden sich auf der linken Seite der Achse. Gelb wird für die Instanzen, die False Positive (FP) sind, verwendet. Das sind Instanzen, die der Klasse falsch zugewiesen wurden. Instanzen, die der Klasse richtig zugewiesen wurden und somit True Positive (TP) sind, erhalten die Farbe Grün. Die gelben und grünen Stacks sind gestapelt und befinden sich auf der rechten Seite der Achse, wobei der gelbe Stack links vom grünen Stack positioniert wird. Das Ergebnis dieser Darstellungsweise ist ein zweiseitiges Histogramm für jede Klasse. Durch diese Darstellung sind Fehler, sowohl FP als auch FN, und korrekte Predictions (TP) gleichermaßen ersichtlich. Dadurch ist die Darstellung unabhängig von bestimmten Metriken und erfüllt somit das Designziel Z2. Auch Designziel Z1 kommt einen Schritt näher, da mit diesen Darstellungen auch Klassen miteinander verglichen werden können.
Zusätzlich wird für jede Seite auf der horizontalen Achse die maximale Anzahl an Instanzen aller Stacks angezeigt. Ohne diese Angabe ist nicht zu sehen, wie viele Instanzen einer Klasse zugeordnet sind, da alle Visualisierungen auf die gleiche Größe skaliert werden. In Abbildung 4.3 ist eine Beispielklasse für diese Darstellungsweise zu sehen.

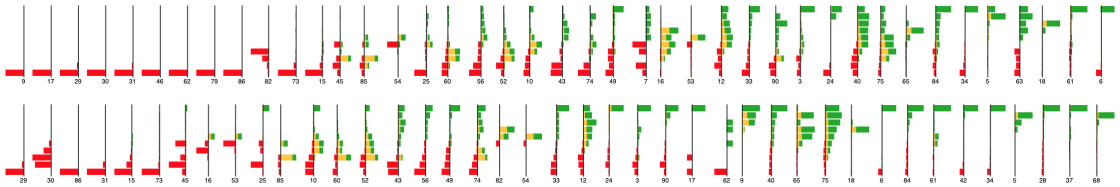

Abbildung 4.4: Auschnitt eines Overviews zweier Modelle, wobei die Klassen nach ihren F-measures geordnet sind.

4.3.2 Gegenüberstellung der Modelle und Overview

Um zwei Modelle vergleichen zu können, werden für jede Klasse die in Kapitel 4.3.1 beschriebenen Visualisierungen erstellt und damit die Modelle gegenübergestellt, wie in Abbildung 5.1 auf dem zweiten Bild zu sehen. Die Visualisierungen für jedes Modell sind unabhängig voneinander scrollbar, da die Klassen sonst zu viel Platz verbrauchen würden. Dadurch können einzelne Klassen einfacher erreicht werden und einzelne Klassen von verschiedenen Modellen miteinander verglichen werden. Aber durch die große Anzahl von Klassen, bleibt es schwierig, die Gesamtperformance eines Modells zu erfassen.
Es wäre von Vorteil, wenn mehr Klassen auf einmal betrachtet werden könnten, um einen Überblick über die Modelle zu bekommen und so schneller Unterschiede zwischen den Modellen ausfindig zu machen. Daher wird zusätzlich ein Overview für jedes Modell erstellt, der die Visualisierungen der Klassen in kleinerer Form gegenüberstellt, wie in Abbildung 4.4 dargestellt. Dabei werden auf die Klassennamen verzichtet, um die Klassen kompakter darstellen zu können. Stattdessen wird jeder Klasse eine Zahl zugewiesen. Wenn die Maus über diese Zahl bewegt wird, erscheint ein Tooltip, der den entsprechenden Klassennamen anzeigt.

4.3.3 Ordnen der Klassen

Auch wenn die Gegenüberstellung einen ersten Eindruck über die Modelle gibt, bleibt der Vergleich und die Feststellung, welches Modell besser performt, weiterhin schwierig. Das liegt an der großen Anzahl an Klassen. Klassen mit guter oder schlechter Performance müssen in der Visualisierung gesucht werden. Um das Suchen zu vereinfachen, wird die Möglichkeit zum Ordnen der Klassen hinzugefügt. Dadurch stehen sich auch ähnliche Klassen, falls es welche gibt, beim Vergleich mit einem anderen Classifier gegenüber und vereinfachen somit den Vergleich. Dafür werden Performance-Metriken benötigt, nach denen die Klassen geordnet werden können. Verwendet werden die Metriken Precision, Recall und F-measure, die für jede Klasse berechnet werden. Das Ordnen der Klassen trägt zur Analyse der Klassenperformance bei, das in Designziel Z1 gefordert ist. In Abbildung 4.4 ist ein Teil eines Overviews zu sehen, auf dem die Klassen nach ihren F-measures geordnet sind.
Das Ordnen nach diesen Metriken liefert nicht immer ein zufriedenstellendes Ergebnis. Deshalb wird noch die Möglichkeit hinzugefügt, die Klassen per Drag & Drop individuell zu sortieren.

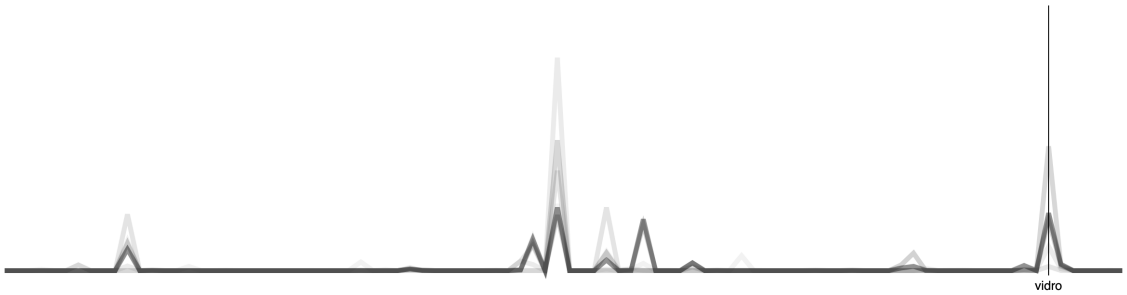

Abbildung 4.5: Parallel-Coordinates aller Instanzen der Klasse "oderoor". Die Polylines zeigen die Prediction-Scores für alle Klassen des Modells. Die Maus befindet über der Klasse "vidro", weshalb die Achse und der Name dieser Klasse eingeblendet wird. Je dunkler die Polyline ist, desto mehr Instanzen fasst sie zusammen.

4.3.4 Between-Class-Confusion

Die Sparklines über den Klassen werden beibehalten, wie in Abbildung 4.3 zu sehen. Sie zeigen die Parallel-Coordinates aller Instanzen in der jeweiligen Klasse an, wobei die Polylines die Prediction-Scores für alle Klassen darstellen. Jedoch geht eine Sparkline nun über die gesamte Breite der Klassendarstellung, da die Modelle eine große Anzahl von Klassen besitzen. Dadurch sind immer noch die Spitzen gut zu erkennen, die auf eine Class-Confusion hinweisen. Hat eine Klasse keine Predictions, wird keine Sparkline hinzugefügt. Durch das Klicken auf einer Sparkline erfolgt die Anzeige einer vergrößerten Darstellung, um analysieren zu können, zu welcher Klasse die Spitze gehört und welche Klassen verwechselt werden. In der vergrößerten Darstellung wird beim Bewegen der Maus über einer Achse diese und der Klassenname eingeblendet, wodurch eine einfache Zuordnung zu einer Klasse möglich ist.
Die Darstellung jeder Instanz als Polyline würde unnötig viel Arbeitsspeicher verbrauchen, weil die Datensätze über 500000 Instanzen beinhalten. Außerdem wären die Parallel Coordinates unübersichtlich, da die Klassen teilweise tausende Instanzen beinhalten und die Verdeckung der Spitzen durch andere Polylines möglich ist. Deshalb werden die Polylines mithilfe des K-Means-Algorithmus in 6 Cluster aufgeteilt. Die Features für jede Instanz sind ihre Prediction-Scores für alle Klassen. Dadurch erfolgt eine Gruppierung der Polylines, die nah beieinander sind. Aus dem Ergebnis des Clusterings erfolgt die Bestimmung einer einzigen Polyline für jeden der 6 Cluster. Dies geschieht durch Berechnung des Mittelwerts für jede Klasse aus den Prediction-Scores aller Instanzen im Cluster. Die Anzahl der Instanzen im Cluster gibt an, wie dunkel die Polyline dargestellt werden soll. Je dunkler die Polyline ist, desto mehr Instanzen fasst sie zusammen. Ein Beispiel hierfür kann in Abbildung 4.5 betrachtet werden. Die Möglichkeit zur Betrachtung der Between-Class-Confusion trägt zur Analyse der Klassenperformance bei, was in Designziel Z1 gefordert ist.

4.3.5 Clustering der Klassen

Die bisher erwähnten Methoden vereinfachen das Vergleichen von ML-Modellen. Jedoch bleibt trotzdem eine große Zahl von Klassen zum Vergleich, die nach und nach analysiert werden müssen, um die Gesamtperformance beurteilen zu können. Um die Anzahl der zu vergleichenden Klassen zu verringern, werden die Klassen mit einem Clustering-Algorithmus gruppiert, und jeder Cluster wird durch einen Repräsentanten dargestellt.

Algorithmus

Als Clustering-Algorithmus wird K-Means verwendet. Dieser hat den Vorteil, das die Anzahl von Clustern vorgegeben werden kann. Wenn ein Algorithmus verwendet werden würde, bei dem die Angabe der Clusteranzahl nicht möglich ist, können für zwei verschiedene Modelle unterschiedliche Anzahlen von Clustern herauskommen, was wiederum den Vergleich erschwert. Außerdem kann bei K-Means die Klasse bestimmt werden, die am nächsten zum Clusterzentrum liegt. Diese Klasse wird dann als ein Repräsentant für das Cluster gesehen, deren Name als Bezeichnung für das Cluster in der Visualisierung verwendet wird.
Beim Clustering soll der Benutzer selber die Anzahl der Cluster festlegen können, um den optimalen Wert für die Visualisierung seines Modells zu finden. Um die Cluster zu bestimmen, werden Features benötigt. Die Berechnung der Features erfolgt aus den Stacks für jede Klasse. Dabei wird für jeden Stack die Anzahl der Instanzen, die er angibt, durch die Gesamtzahl der Instanzen in der Klasse geteilt. Die Gesamtzahl besteht aus der Summe aller Stacks der Klasse. Dadurch wird der prozentuale Anteil jedes Stacks an die Gesamtzahl der Instanzen in der Klasse wiedergegeben. Folglich werden für jede Klasse 30 Features berechnet, die zusammen eine 92×30 Matrix ergeben. Diese Matrix dient als Eingabe für den Algorithmus.

Darstellung des Clustering-Ergebnisses

Sind die Klassen geclustert, stellt sich die Frage, wie die einzelnen Cluster dargestellt werden sollen. Um die Anzahl der zu vergleichenden Darstellungen gering zu halten, erfolgt die Darstellung jedes Clusters durch eine einzige Visualisierung. Hierfür wird auf die vorherige Visualisierung der Klassen zurückgegriffen, die für die Darstellung der Cluster angepasst wird. Dazu werden aus den Klassen im Cluster 30 neue Stacks erstellt, deren Bestimmung anhand eines Beispiels erklärt wird. Seien in einem Cluster 6 Klassen. Betrachtet wird ein bestimmter Prediction-Score-Bereich (z.B. $0.8 - 0.9$) für eine der Stackfarben, wie z.B. für die grünen Stacks. Die Anzahl der Instanzen jeder Klasse in diesem Bereich seien

$$S = \{\ 12, 65, 86, 146, 890, 1135\ \}.$$

Das Maximum aus diesen Werten (hier 1135) gibt die Anzahl der Instanzen für den neuen Stack an. Um die Werte der anderen Stacks darzustellen, wird über den neuen Stack ein Boxplot gelegt. Die Werte für die Erstellung des Boxplots sind die der Menge S. Dadurch ist es möglich, die Verteilung dieser Werte übersichtlich darzustellen. Durch die Anwendung dieses Vorgangs für alle 30 Stackbereiche, erfolgt die Erstellung der Clusterdarstellung. Hat keine der Klassen in einem Bereich Instanzen, wird auch kein Stack erstellt. Zusätzlich befindet sich über der Darstellung des Clusters die Anzahl der Klassen im Cluster. Ein Beispiel für diese Clusterdarstellung ist in Abbildung 4.6 zu sehen.

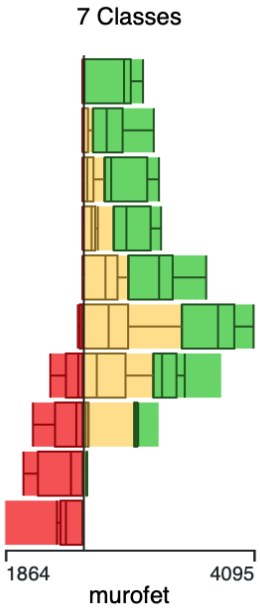

Abbildung 4.6: Darstellung eines Clusters mit 7 Klassen. Die Klasse "murofet" liegt am nächsten zum Clusterzentrum und stellt somit den Repräsentanten für diesen Cluster dar. Die Boxplots an jedem Stack geben die Verteilungen aller Klassen im Cluster für den jeweiligen Stackbereich wieder.

Interaktionen mit den Clusterdarstellungen

Um die Clusteringergebnisse untersuchen zu können, werden Interaktionsmöglichkeiten zu den Darstellungen der Cluster hinzugefügt. Durch das Klicken auf einen Cluster werden die darin enthaltenen Klassen dargestellt. Sind zwei Modelle gegenübergestellt, kann durch das Klicken auf eine angezeigte Klasse die entsprechende Klasse und der damit verbundene Cluster im anderen Modell angezeigt werden. Für das obere Modell befinden sich die Klassen darunter und für das untere Modell darüber, wie in Abbildung 4.7 zu sehen. Außerdem werden die aktiven Cluster Klassen umrandet. Durch diese Interaktionen ist der direkte Zugang zu den Klassen von den Clusterdarstellungen aus gegeben und vereinfacht den Vergleich der Ergebnisse zweier Modelle.
Durch das Bewegen der Maus über der Angabe, wie viele Klassen im Cluster enthalten sind, erscheint ein Tooltip, der die Namen der im Cluster enthaltenen Klassen anzeigt. Dies ist auch in Abbildung 4.7 zu sehen.

Overview des Clusteringergebnisses

Wenn die Klassen geclustert sind, erfolgt eine Anpassung des Overviews. Dazu wird jeder Cluster in einer neuen Zeile angezeigt. Zusätzlich befindet sich über jedem Cluster der Name des Clusterrepräsentanten, was eine Zuordnung zu den Clusterdarstellungen ermöglicht. Sind zwei Modelle hinzugefügt, werden die Ergebnisse beider Modelle nebeneinander angezeigt. Links das Ergebnis des oberen Modells und rechts das des unteren Modells. Der Overview ermöglicht die Betrachtung und den Vergleich mehrerer Cluster auf einmal und gibt so einen Überblick über die

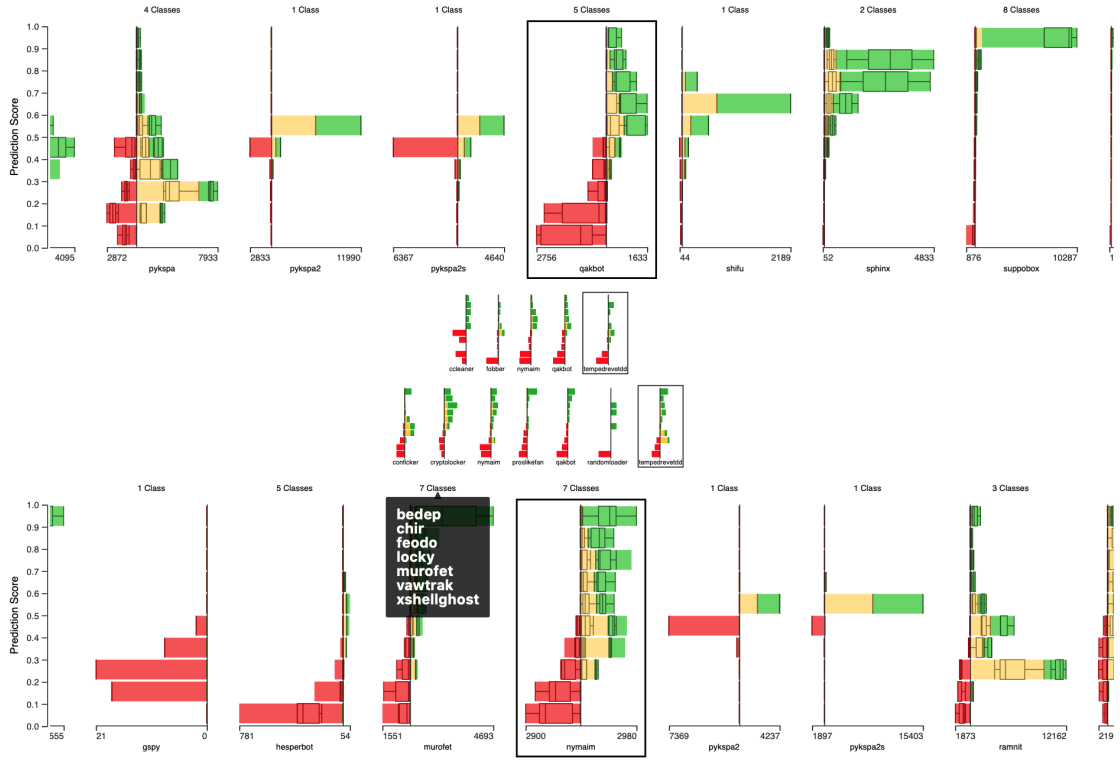

Abbildung 4.7: Interaktionen mit den Clusterdarstellungen. Unter bzw. über dem Modell erscheinen die Klassen, die im Cluster sind. Es wurde auf die umrandete Klasse geklickt, wodurch diese auch im Cluster des anderen Modells angezeigt wird. Beim Bewegen der Maus über der Angabe, wie viele Klassen im Cluster sind, erscheint ein Tooltip mit den entsprechenden Klassennamen.

Ergebnisse des Clusterings. Außerdem kann durch das Klicken auf eine Klasse, wenn zwei Modelle dargestellt sind, die entsprechende Klasse im Cluster des anderen Modells auf die gleiche Ebene wie die Ebene der geklickten Klasse gestellt werden, was in Abbildung 4.8 demonstriert ist. Dadurch ist es möglich, die Ergebnisse einfacher und schneller zu vergleichen, da die Suche der Klasse im anderen Modell entfällt und die Cluster auf einer Ebene sind.

Ordnen von Cluster

Auch das Ordnen wird weiterhin unterstützt. Hierfür wird für jeden Cluster der Durchschnitt aus allen darin enthaltenen Klassen für die entsprechende Metrik berechnet, nach welcher die Cluster dann geordnet werden. Dies ist sowohl in der Darstellung mit den Boxplots als auch im Overview möglich. Die Clusterdarstellungen können wie die Klassendarstellungen per Drag & Drop umpositioniert werden.

Das Clustering der Klassen, die Darstellung der Clusteringergebnisse sowie die Interaktionen mit den Clustern erlauben die Analyse der Gesamtperformance eines Classifiers, was in Designziel Z1 gefordert ist.

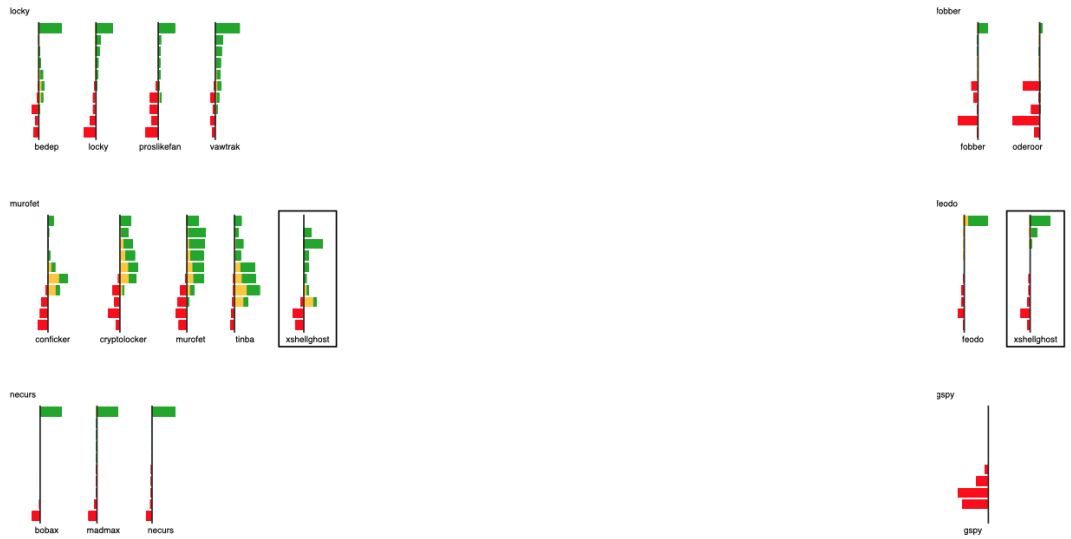

Abbildung 4.8: Ausschnitt aus dem Overview der Clusteringergebnisse zweier Modelle. Links befindet sich die Darstellung des einen Modells und rechts das des anderen. Die Klasse "xshellghost" wurde angeklickt, weshalb die gleiche Klasse im Cluster des anderen Modells angezeigt wird.

4.3.6 Verbindung von Performance und Daten

Durch das Doppelklicken auf einen Stack werden die entsprechenden Instanzen in einer Tabelle angezeigt, wie in Abbildung 4.9 zu sehen. Dies ist sowohl in der Darstellung der Klassen als auch in der Darstellung der Cluster möglich. Im Gegensatz zu Squares wird auf die Anzeige von Polylines verzichtet, da dies bei 92 Klassen unübersichtlich wäre. Auch wenn die Klassen geclustert sind, ist der Informationsgewinn aus den Polylines schwierig, da die Scores einer Instanz für die Klassen in einem Cluster unterschiedlich hoch ausfallen können. Zusätzlich ist durch das Anklicken einer Klasse im Overview ohne Clustering das Anzeigen aller Instanzen in dieser Klasse möglich. Durch diese Interaktionsmöglichkeiten wird das Designziel Z3 erreicht, wobei es um den direkten Zugang von der Performance zu den Daten ging.
Die Tabelle zeigt für jede Instanz im Stack den Namen der Domain, das Truth, den Truth-Score, die Prediction sowie den Prediction-Score an. Das Truth ist die Klasse, zu der die Instanz in Wirklichkeit gehört und der Truth-Score ist der Prediction-Score der Instanz für diese Klasse. Sind zwei Modelle hinzugefügt, zeigt die Tabelle die entsprechenden Daten für beide Modelle an. Die Tabelle kann mithilfe des Suchfeldes gefiltert sowie jede Spalte auf- und absteigend sortiert werden. Dies ermöglicht die Analyse auf der Instanzebene und führt zur Erfüllung des Designziels Z1, da nun die Analyse in allen 3 Ebenen durchführbar ist.

Search:

		rnn_mcc_ds0_full.csv			cnn_mcc_ds0_full.csv		
Domain	**Truth**	**Truth-Score**	**Prediction**	**Prediction-Score**	**Truth-Score**	**Prediction**	**Prediction-Score**
abolgtcxynut3.com	bedep	0.0069	corebot	0.9334	0.9341	bedep	0.9341
accpxepcbhhobvgc.com	bedep	0.0284	sphinx	0.8910	0.0293	sphinx	0.8683
adlnefevxnhbcfvp.com	bedep	0.0293	sphinx	0.8754	0.0251	sphinx	0.8567
ahfewrdaysaugaow.com	bedep	0.0260	sphinx	0.9051	0.0347	sphinx	0.8100
ailflniedwad.com	bedep	0.0759	ramnit	0.3122	0.1400	tinba	0.2097
aixseaqaraqfvxhr.com	bedep	0.0331	sphinx	0.8648	0.0311	sphinx	0.8558
ambiwqvmhdfdkpgb.com	bedep	0.0278	sphinx	0.8847	0.0359	sphinx	0.8602
aottvkumoupk.com	bedep	0.0995	ramnit	0.2577	0.0824	tinba	0.4315

Showing 1 to 9 of 651 entries

Abbildung 4.9: Darstellung der Instanzen in einer Tabelle für zwei Modelle, wenn ein Stack angeklickt wird. Angezeigt sind die Domain, das Truth und für jedes Modell der Truth-Score, die Prediction sowie der Prediction-Score. Das Suchfeld ermöglicht die Filterung der Daten. Die Spalten lassen sich auf- und absteigend sortieren.

5 Implementierung

Für die Entwicklung eines Prototypen wurde auf die Kombination aus einem Python-Websocket-Server (Python-Version 3.7) und einem Javascript-Client (Javascript-Version ECMAScript6+) gesetzt. Zusätzlich wurde für die Gestaltung der Oberfläche des Clients das Bootstrap-Framework [Mar] und für die Erstellung der Visualisierungen die Library D3.js [Bos] verwendet.

5.1 Front-End: Javascript-Client

Das Front-End bildet ein Javascript-Client, der im Browser läuft. Die Kommunikation mit dem Server erfolgt über das Websocket-Protokoll. Dieses Protokoll erlaubt den bidirektionalen Austausch von Nachrichten und ermöglicht so eine einfache Kommunikation [FM11].
Wird eine Datei hinzugefügt, sendet der Client diese an den Server. Das Senden der Datei erfolgt dabei in kleineren Paketen, um den Speicherverbrauch des Browsers gering zu halten. Dadurch muss der Client nicht die großen Datensätze selbst im Arbeitsspeicher halten, sondern kann die Daten auf den Server auslagern. Außerdem wird verhindert, das die Bedienoberfläche bei längeren Berechnungen nicht reagiert, da die rechenintensiven Berechnungen der Server übernimmt. Der Client hat weiterhin Zugriff auf den Datensatz und muss nur eine Anfrage senden, um die benötigten Daten zu erhalten. Beim Klicken auf einen Stack wird daher eine Anfrage an den Server geschickt, um die damit verbundenen Daten für die Tabelle zu erhalten. Die für das Clustering benötigte Featurematrix wird vom Client berechnet, da dies nur einfache Rechenoperationen wie Addition und Division erfordert. Außerdem ist die Featurematrix mit der Dimension 92×30 relativ klein und die benötigten Daten für die Berechnungen sind vorhanden.
Die Eingabe der Daten erfolgt als CSV-Datei, die die folgenden 95 Spalten enthält:

Domain	**Prediction**	**Truth**	**0**	**...**	**91**
Name der Domain	Prediction für die Instanz	Klasse, zu der die Instanz in Wirklichkeit gehört	Prediction-Score der Instanz für die Klasse 0	...	Prediction-Score der Instanz für die Klasse 91

5.2 Back-End: Python-Websocket-Server

Das Back-End stellt ein Python-Websocket-Server dar, der die aufwendigen Berechnungen für den Client ausführt. Der Server erhält vom Client den Datensatz des Modells. Basierend auf diesen Datensatz berechnet er die für die Visualisierungen benötigten Daten und sendet sie an den Client. Für das Clustering wird der K-Means-Algorithmus aus der "scikit-learn" -Bibliothek [sci] verwendet.

Die Clusteranzahl und die Features erhält der Server vom Client. Zusätzlich wird ein Random-State gesetzt. Dies sorgt dafür, dass der Zufall deterministisch wird und so nicht unterschiedliche Ergebnisse geliefert werden, wenn die Berechnung für dieselbe Clusteranzahl wiederholt wird.
Die Tabelle wird mit dem DataTables-Plugin [Dat] für Javascript erstellt und ermöglicht das Scrollen durch alle Instanzen des angeklickten Stacks. Da nur die Instanzen entsprechend zur Scrollposition gerendert werden, ist der Speicherverbrauch gering und ermöglicht eine schnelle Exploration der Instanzen.
Die Aufteilung in Front-End und Back-End hat den Vorteil, dass das Back-End auf einen Rechner mit genügend Leistung für größere Datensätze oder für Datensätze mit noch größerer Anzahl an Klassen ausgelagert werden kann. Somit ist eine gute Skalierbarkeit gegeben.

5.3 Grafische Benutzeroberfläche

Die Benutzeroberfläche ist in zwei Tabs aufgeteilt, die im Folgenden beschrieben werden:

Data Upload Der erste Tab dient zum Upload der Dateien, die zum Websocket-Server gesendet werden, und ist in Abbildung 5.1 auf dem ersten Bild zu sehen. Das Hinzufügen einer Datei ermöglicht die Untersuchung eines Modells. Durch die Auswahl von zwei Dateien, können zwei Modelle untersucht und verglichen werden. Durch die Auswahl einer neuen Datei ist die Ersetzung des angezeigten Modells möglich. Außerdem kann das als Zweites hinzugefügte Modell wieder entfernt werden, wenn nur ein Modell untersucht werden soll.

Visualization Der zweite Tab dient zur Anzeige der Visualisierungen, die im vorherigen Kapitel erklärt wurden. Die Oberfläche des zweiten Tabs ist in Abbildung 5.1 auf dem zweiten Bild dargestellt. Im oberen Bereich ganz links befindet sich ein Auswahlmenü, mit dem bestimmt werden kann, nach welcher Metrik die Klassen sortiert werden sollen. Rechts davon befinden sich zwei Radiobuttons, die es ermöglichen, die Klassen für die ausgewählte Metrik auf- oder absteigend zu sortieren. Wenn die Klassen mit dem Drag & Drop-Feature umsortiert wurden, kann mit dem Reload-Order-Button die ausgewählte Metrik neu geladen und somit die vorgenommenen Veränderungen zurückgesetzt werden. Ganz rechts ist der Clustering-Button platziert, mit dem das Clustering aktiviert wird. Unter diesen Elemente ist das Modell dargestellt, das in der oberen File-Auswahl hinzugefügt wurde. Das zweite Modell, das in der unteren File-Auswahl ausgewählt wurde, wird unter dem Ersten platziert. Zusätzlich wird für jedes Modell der Name der Datei und die Accuracy hinzugefügt. Zwischen diesen Modellen befindet sich der Overview, der ein- und ausgeklappt werden kann.
Wenn auf den Clustering-Button geklickt wird, ist die in Abbildung 5.2 dargestellt Oberfläche zu sehen. Neben dem Clustering-Button erscheint dann ein Bedienelement, mit dem die Anzahl der Cluster festgelegt werden kann. Der Overview verschiebt sich unter das zweite Modell, da dieser in der vorherigen Position beim Vergleichen der Cluster stören würde.

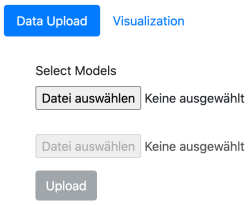

Abbildung 5.1: Die grafischen Benutzeroberflächen des Prototypen. Das erste Bild zeigt den ersten Tab, in dem die Dateien der Modelle hinzugefügt werden können. Auf dem zweiten Bild ist der Visualization-Tab zu sehen, der die Visualisierungen der Modelle anzeigt.

Abbildung 5.2: Die Grafische Benutzeroberfläche, wenn die Klassen geclustert sind. Die Anzahl der Cluster beträgt 20 und ist für zwei Modelle angezeigt.

6 Evaluation

Dieses Kapitel befasst sich mit der Evaluation des vorgestellten Konzepts und des entwickelten Prototypen. Im Squares-Paper [RAL+17] sind 6 Aufgaben erstellt worden, die die häufigen Anforderungen an einer Performance-Analyse von ML-Modellen repräsentieren. Diese Aufgaben sind aus den Ergebnissen einer Umfrage erstellt worden. Dabei ging es um Methoden und Schwierigkeiten, die einem bei der Entwicklung von Multiclass-Classifiern begegnen. Durchgeführt wurde die Umfrage in großen Technologieunternehmen, wobei 102 Fachleute für Machine Learning mitgemacht haben. Deshalb ist die Verwendung dieser Aufgaben für die Evaluation sinnvoll.
Die erstellten Aufgaben lauten wie folgt:

Aufgabe 1 Welcher Classifier hat die größere Fehleranzahl?

Aufgabe 2 Welche Klasse hat die meisten Fehler?

Aufgabe 3 Auswahl einer Instanz, die vom Classifier falsch zugeordnet wurde, mit einem Prediction-Score größer gleich 0.9.

Aufgabe 4 Welcher Classifier hat die schlechtere Verteilung?

Aufgabe 5 Welche Klasse hat die schlechteste Verteilung?

Aufgabe 6 Welche zwei Klassen werden am meisten miteinander verwechselt?

Aufgabe 1 ist zur Analyse der Gesamtperformance gedacht, Aufgabe 2 zur Analyse der Klassenperformance und Aufgabe 3 zur Analyse der Performance auf Instanzebene. Die Aufgaben 4 und 5 konzentrieren sich auf die Analyse der Performance anhand des Prediction-Scores. Aufgabe 6 bezieht sich auf die Between-Class-Confusion.
Die Evaluation bei Squares erfolgte durch eine Studie, in der von Probanden Squares mit einer Confusion Matrix verglichen wurde. Dazu erfolgte die Durchführung der erwähnten Aufgaben sowohl mit einer Confusion Matrix als auch mit Squares. Dabei wurde die Zeit gemessen, die für die Erfüllung der Aufgabe benötigt wurde und die Probanden wurden befragt, wie hilfreich sie die jeweilige Visualisierung fanden. Das Ergebnis dieser Studie ist, dass mit Squares die Analyse schneller und genauer durchgeführt werden kann als mit einer Confusion Matrix und Squares von vielen Probanden bevorzugt wird.
Die Evaluation des vorgestellten Konzepts erfolgt anhand der Durchführung der erwähnten Aufgaben. Dazu wird ein Classifier basierend auf einem CNN und ein Classifier basierend auf einem RNN verwendet. Beide Classifier ordnen die Instanzen in 92 Klassen ein. Der Testdatensatz, der für die Aufgaben herangezogen wird, beinhaltet knapp 550000 Instanzen, worauf die Classifier trainiert sind. Die Aufgaben werden nacheinander behandelt, um zu überprüfen, ob das Konzept und der Prototyp funktionieren und in welchen Bereichen die Schwächen liegen.

6.1 Durchführung der Aufgaben

6.1.1 Aufgabe 1

Bei dieser Aufgabe geht es um die Bestimmung des Classifiers mit der größeren Anzahl an Fehlern. Hierzu werden zuerst die Klassen der Classifier nach ihren F-measures aufsteigend sortiert und der Overview betrachtet. Das Ergebnis ist in Abbildung 4.4 zu sehen. Oben ist der Overview des CNN und unten der Overview des RNN abgebildet. Darauf ist zu erkennen, dass beim CNN der Anteil der roten und gelben Stacks, die die Fehler anzeigen, im unteren Bereich des F-measures höher ist, was auf eine höhere Fehleranzahl hinweisen könnte. Durch das Verändern der Clusteranzahl und die Betrachtung der Ergebnisse im Overview als auch durch Anklicken der Cluster, wird eine Clusteranzahl von 12 für beide Classifier festgestellt, die für den Vergleich geeignet sind. Dabei wurden nach jeder Änderung der Clusteranzahl die Klassen in den Clustern näher betrachtet, um festzustellen, ob die Klassen sich auch ähnlich und ob nicht passende Klassen enthalten sind. Zusätzlich sind die Cluster nach ihren F-measures aufsteigend sortiert und per Drag & Drop angepasst. Das Ergebnis dieses Vorgangs ist in Abbildung 6.1 abgebildet. Auch hier ist der Anteil von roten und gelben Stacks beim CNN höher. Jedoch ist es schwierig aus diesen Darstellungen verlässlich eine Fehleranzahl abzulesen, da alle Klassen und Cluster auf die gleiche Größe skaliert sind. Um ungefähr eine Anzahl abzuschätzen, können die Angaben zu den Anzahlen der Instanzen unter den Stacks betrachtet werden. Dann müsste für jedes Cluster anhand dieser Zahlen ungefähr eine Fehleranzahl abgeschätzt und addiert werden, um die Fehleranzahl für den gesamten Classifier zu erhalten. Jedoch ist dieser Vorgang sehr zeitaufwendig und vereinfacht nicht die Analyse des Classifiers. Somit ist die Erfüllung dieser Aufgabe anhand der Visualisierungen schwierig. Wenn durch die Visualisierung der Unterschied der Fehleranzahl nicht deutlich wird, kann die Accuracy des Classifiers zu Rate gezogen werden. Der CNN hat eine Accuracy von etwa 0.84 und der RNN von etwa 0.86. Folglich ist die Fehleranzahl beim CNN etwas höher als beim RNN.
Es ist aber zu erwähnen, dass die Fehleranzahlen beider Classifier nicht weit auseinander liegen. Beim CNN liegt der Wert bei etwa 85000 und beim RNN bei etwa 77000 liegt. Bei größerer Differenz der Fehleranzahl wäre der Unterschied der Anteile roter und gelber Stacks größer und eine Abschätzung somit einfacher.

6.1.2 Aufgabe 2

Die Klasse mit den meisten Fehlern soll in dieser Aufgabe bestimmt werden. Dafür wird der CNN-Classifier betrachtet. Zuerst erfolgt das Ordnen der Klassen nach einer Metrik. Welche Metrik ausgewählt wird, hängt davon ab, welche Fehler betrachtet werden sollen. Sind nur die Instanzen einer Klasse von Interesse, die der Klasse falsch zugeordnet wurden (FP), dann werden die Klassen nach der Precision geordnet. Sind auch die Instanzen von Interesse, die einer Klasse nicht zugeordnet wurden, obwohl sie zu ihr gehören, erfolgt die Ordnung nach dem Recall oder dem F-measure. Im Falle des CNN-Classifiers spielt dies keine Rolle, da es hier 9 Klassen gibt, die bei allen Metriken den Wert 0 haben, wie in Abbildung 6.2 zu sehen. Darauf ist der Overview des Classifiers abgebildet, auf dem die Klassen aufsteigend nach dem Recall sortiert sind. Folglich haben diese 9 Klassen die meisten Fehler, da ihnen gar keine Instanz richtig zugeordnet wurde.

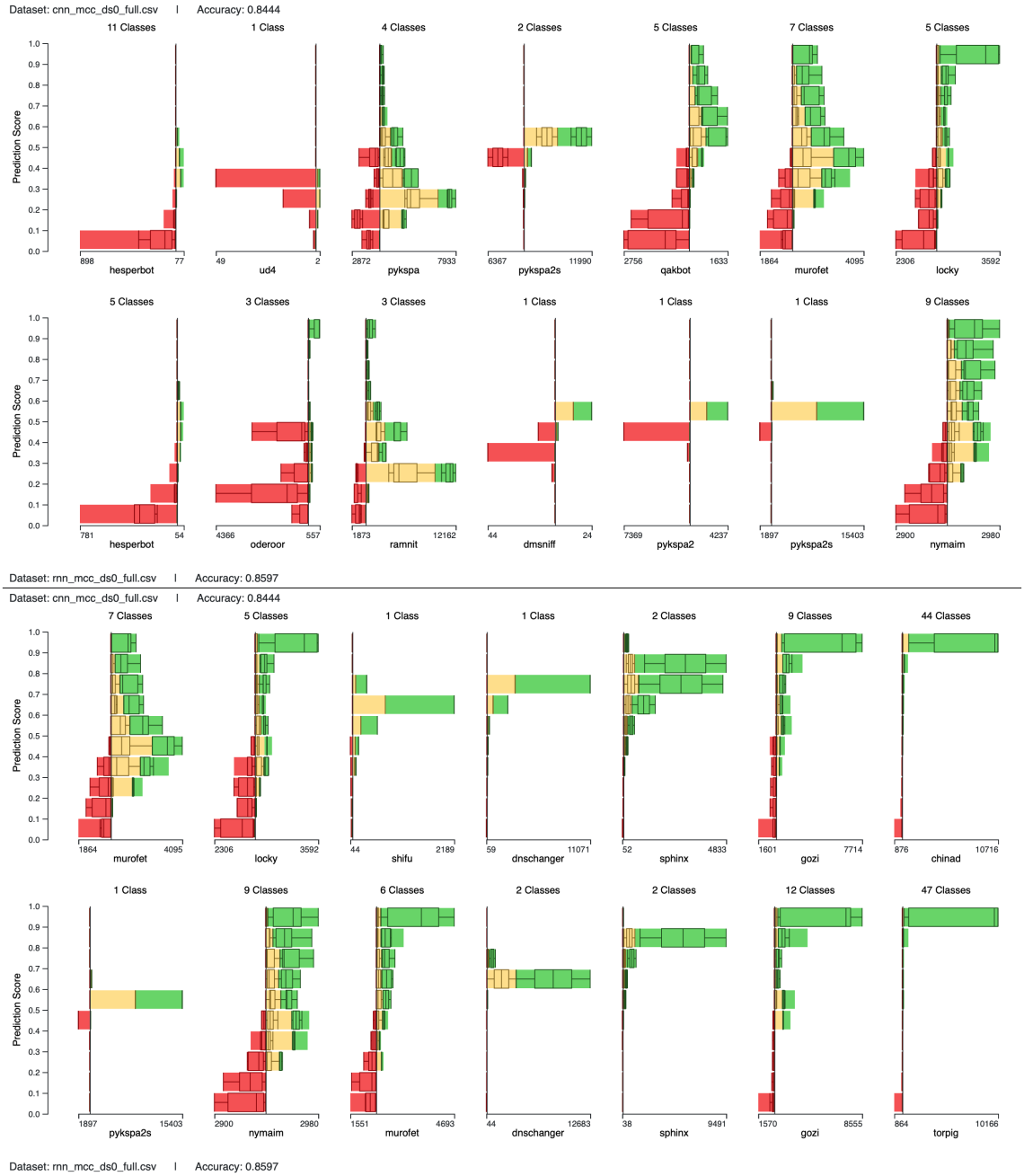

Abbildung 6.1: Die Classifier RNN und CNN in 12 Cluster eingeteilt und aufsteigend nach den F-measures sortiert, wobei per Drag & Drop die Sortierung angepasst wurde. Die letzten zwei Cluster in der oberen Darstellung wiederholen sich in der unteren.

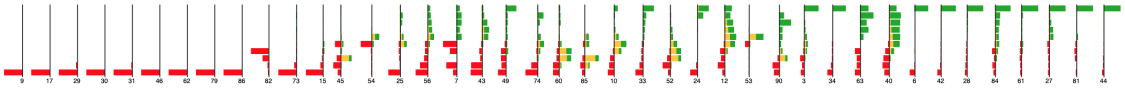

Abbildung 6.2: Teil des Overviews des CNNs, auf dem die Klassen nach dem Recall aufsteigend geordnet sind.

6.1.3 Aufgabe 3

In dieser Aufgabe ist die Bestimmung eines Fehlers, dessen Prediction-Score größer gleich 0.9 ist, gefordert. Verwendet wird wieder der CNN-Classifier. Um diese Aufgabe zu erfüllen, muss bei den Klassen Ausschau nach einem gelben Stack im Score-Bereich von 0.9 bis 1.0 ausgehalten werden. Ist so ein Stack gefunden, können durch Anklicken des Stacks die darin enthaltenen Instanzen in der Tabelle angezeigt werden. Dabei fällt das Problem auf, dass einzelne Stacks sehr schmal sein können, was das Anklicken erschwert oder gar nicht möglich ist. Deshalb kann alternativ die Klasse im Overview angeklickt werden, um alle darin enthaltenen Instanzen anzuzeigen. Durch die Sortierung der Instanzen nach den Prediction-Scores oder durch das Filtern der Daten mit dem Suchfeld können entsprechende Instanzen gefunden werden, falls vorhanden.

6.1.4 Aufgabe 4

Der Classifier mit der schlechteren Verteilung ist gesucht. Es werden wieder der CNN-Classifier und der RNN-Classifier verwendet. Zur Erfüllung der Aufgabe erfolgt zuerst das Clustern der Klassen beider Classifier, wieder mit einer Clusteranzahl von 12. Die Cluster sind nach ihren F-measures aufsteigend sortiert und per Drag & Drop angepasst, damit ähnliche Cluster sich gegenüberstehen. Das Ergebnis ist in Abbildung 6.1 dargestellt.
Werden die zwei ersten Cluster beider Classifier miteinander verglichen, fällt auf, dass das CNN in den beiden Cluster mehr Klassen zusammengefasst hat als das RNN. An den Boxplots der Cluster "hesperbot" beider Classifier ist zu erkennen, dass die Klassen in den entsprechenden Bereichen in etwa gleich verteilt sind. Somit hat bisher das RNN aufgrund der niedrigeren Klassenanzahl eine bessere Verteilung. Als nächstes folgt die Betrachtung des Clusters "pykspa" des CNNs und "ramnit" des RNNs. Das CNN hat hier einen größeren Anteil an roten Stacks, wobei sie beim CNN auch weiter oben vorkommen. An den Boxplots ist auch zu sehen, dass bei den meisten Klassen die roten Stacks etwa gleich viele Instanzen enthalten. Der Cluster "ramnit" hat im Score-Bereich von über 0.9 auch einen größeren Anteil von grünen Stacks. Auf diese Weise wird der Vergleich für alle Cluster durchgeführt. Am Ende angekommen, fällt bei den letzten zwei Clustern (die Cluster "gozi" beider Classifier, "chinad" beim CNN und "torpig" beim RNN) auf, dass diese bei beiden Classifiern ähnlich sind, aber das RNN hat mehr Klassen in diesen Clustern. Daraus folgt, dass die Verteilung des RNNs besser ist als die des CNNs. Folglich hat das CNN die schlechtere Verteilung.

6.1.5 Aufgabe 5

Bei dieser Aufgabe soll die Klasse mit der schlechtesten Verteilung bestimmt werden. Dazu wird wieder der CNN-Classifier betrachtet. Zur Bestimmung der schlechtesten Verteilung erfolgt zuerst das Ordnen der Klassen nach dem Recall oder dem F-measure. Die Precision ist nicht dafür geeignet, da die Instanzen, die FN sind, nicht berücksichtigt werden. Das Ergebnis ist in Abbildung 6.2 zu sehen. Auf dem Overview ist zu erkennen, dass die ersten 9 Klassen die schlechtesten Verteilungen haben. Außerdem ist auch erkennbar, dass die Verteilung immer besser wird, je weiter rechts die Klasse liegt.

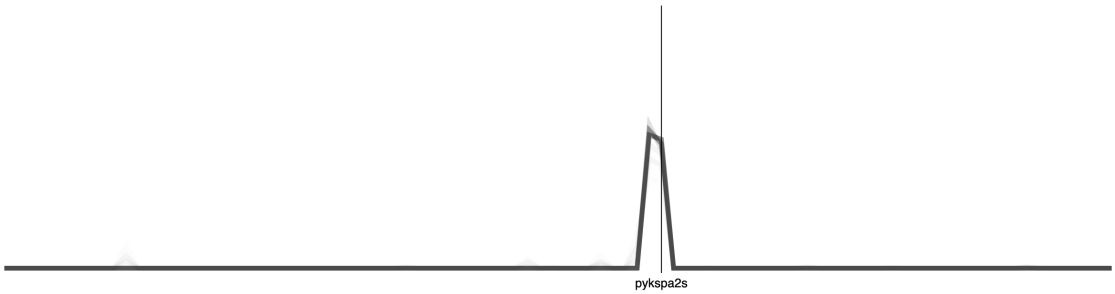

Abbildung 6.3: Parallel Coordinates der Klasse "pykspa2". Diese Klasse wird oft mit der Klasse "pykspa2s" verwechselt.

6.1.6 Aufgabe 6

Nun sollen die zwei Klassen bestimmt werden, die am meisten miteinander verwechselt werden. Verwendet wird hier das CNN. Um die zwei Klassen zu bestimmen, erfolgt die Betrachtung der Sparklines über den Klassen. Dabei ist es hilfreich, die Klassen nach einer Performance-Metrik zu sortieren und die Klassen mit den niedrigen Werten anzuschauen. Denn je höher der Wert ist, desto besser ist die Performance und desto weniger Verwechslungen kommen vor. Bei der Analyse sind auf etwa gleich hohe Spitzen zu achten, da dann die Prediction-Scores der anderen Klasse ähnlich hoch sind und somit eine Verwechslung auftreten kann.
Bei der Analyse fallen die Klassen "pykspa2" und "pykspa2s" auf. Die Parallel Coordinates der Klasse "pykspa2" ist in Abbildung 6.3 zu sehen. Diese Klasse kann mit der Klasse "pykspa2s" verwechselt werden. Um festzustellen, ob dies der Fall ist, werden die Instanzen dieser Klasse in der Tabelle näher betrachtet. Dazu wird die entsprechende Klasse im Overview angeklickt, um alle Instanzen der Klasse anzuzeigen. Dabei fällt auf, dass zwischen diesen Klassen oft Verwechslungen auftreten.
Zwei weitere Klassen, die miteinander verwechselt werden können, sind "oderoor" und "vidro". Die Parallel Coordinates der Klasse "oderoor" sind in Abbildung 4.5 zu sehen. Bei der Betrachtung fällt eine mögliche Verwechslung mit der Klasse "vidro" auf. Durch nähere Analyse der Instanzen in der Tabelle bestätigt sich diese Vermutung. Dieses Vorgehen wird für alle Klassen, bei denen ähnlich hohe Spizen auffallen, wiederholt.
Zur Feststellung, welche Klassen am meisten miteinander verwechselt werden, wird darauf geachtet, wie dunkel die Linien sind, die die Spitzen bilden. Diese Linien sind bei der Klasse "pykspa2" dunkler. Daraus folgt, dass die Linie bei "pykspa2" mehr Instanzen aus der Gesamtzahl der Instanzen der Klasse zusammengefasst hat als die Klasse "oderoor". Das Ergebnis ist folglich, dass die Klassen "pykspa2" und "pykspa2s" am meisten miteinander verwechselt werden.

6.2 Fazit

Die Durchführung der Aufgaben hat gezeigt, dass das entwickelte Konzept funktioniert. Die meisten Aufgaben konnten problemlos durchgeführt werden. Jedoch gibt es noch einige Schwächen, die bei den Analysen aufgefallen sind. Eine dieser Schwächen ist das Abschätzen der Fehleranzahl für den

Classifier anhand der Visualisierung, da die Klassen auf die gleiche Größe skaliert sind. Außerdem erschwert diese Sklalierung auch das Anklicken kleinerer Stacks, weshalb der Umweg über den Overview genommen werden muss.

7 Zusammenfassung und Ausblick

Das Ziel dieser Arbeit bestand in der Entwicklung eines Analysesystems, das den Vergleich verschiedener ML-Modelle anhand von Visualisierungen ermöglicht. Als Basis für das System wurde Squares [RAL+17] verwendet und für die ML-Modelle, die DGAs klassifizieren, angepasst. Das entwickelte Konzept in dieser Arbeit ermöglicht den Vergleich von Classifiern mit großer Klassenanzahl. Dies wird durch das Clustering der Klassen erreicht, wodurch Klassen mit ähnlicher Performance gruppiert werden. Die Darstellung der Cluster als Histogramme in Kombination mit Boxplots vereinfacht den Vergleich der Classifier. Außerdem erlauben geeignete Interaktionen mit den Clustern das Untersuchen der Ergebnisse des Clusterings.
Durch die Visualisierung einzelner Klassen als Histogramme ist ihr Vergleich möglich. Die Klassen sowie Cluster können sortiert werden, um eine Übersicht über sie zu bekommen und beim Vergleich Klassen oder Cluster mit ähnlicher Performance gegenüber zu haben. Ein Überblick über die Klassen wird durch das zusätzliche Anzeigen der Klassen in kleinerer Form ermöglicht. Auch für das Clustering ist ein Überblick vorhanden, der den Vergleich der Ergebnisse des Clusterings vereinfacht.
Mit den Sparklines über den Klassendarstellungen können Between-Class-Confusions ermittelt werden. Die Sparklines zeigen dafür die Parallel Coordinates einer Klasse für alle Instanzen, die dieser Klasse zuordnet wurden, an. Für jede Instanz wird dabei eine geclusterte Polyline mit den gemittelten Prediction-Scores für alle Klassen erstellt. Die Sparklines können durch Anklicken vergrößert werden, um sie genauer zu untersuchen.
Das Anklicken von Stacks und Klassen im Overview zeigt eine Tabelle mit den darin enthaltenen Instanzen an. Dies ermöglicht den direkten Zugang von der Visualisierung zu den Daten. Die Instanzen in dieser Tabelle können nach den Spalten sortiert und mit einem Suchfeld gefiltert werden.
Auf Basis dieses Konzepts wurde ein Prototyp erstellt, mit dem das Konzept ausgewertet wurde. Die Evaluation erfolgte durch die Durchführung von Aufgaben, die die gängigen Vorgänge bei der Analyse eines ML-Modells repräsentieren. Die Ergebnisse der Evaluation zeigen, dass mit dem entwickelten Konzept die meisten Aufgaben problemlos durchgeführt werden können. Jedoch zeigten sich auch dabei Probleme, wie das Abschätzen der Fehleranzahl für Classifier und das Anklicken von kleinen Stacks.

Ausblick

Ein Problem beim Konzept ist, dass die Bestimmung der Fehleranzahl eines Classifiers schwierig ist, wenn die Klassen geclustert sind. Hierfür ist die Anpassung der Visualisierungen der Cluster erforderlich. Dazu könnten statt auf den Stacks auf Strips gesetzt werden, die eine bestimmte Anzahl an Instanzen zusammenfassen. Dadurch wäre auch das Problem mit dem Anklicken von kleinen Stacks gelöst. Alternativ können auch Boxen für bestimmte Prediction-Score-Bereiche angezeigt

werden, um einzelne Instanzen anklicken zu können.
Beim Clustering wurde auf K-Means gesetzt, um beim Vergleich der Classifier gleich viele Cluster zu haben. Aktuell ist es für die Bestimmung einer zufriedenstellenden Clusteranzahl nötig, verschiedene Anzahlen auszuprobieren. Eine Automatisierung dieses Prozesses wäre wünschenswert, um den Vergleich zu vereinfachen. Alternativ kann getestet werden, ob eine Clustering-Methode, die keine Vorgabe der Clusteranzahl benötigt, bessere Ergebnisse liefert und ein Vergleich der Ergebnisse trotzdem möglich ist.
Bei der Analyse der Instanzen werden nur die Domain, die Prediction, das Truth sowie die Scores für die Prediction und das Truth in der Tabelle angezeigt. Das Hinzufügen der Features zu der Tabelle würde eine genauere Analyse der Instanzen ermöglichen.

Literaturverzeichnis

[ACD+15] S. Amershi, D. M. Chickering, S. M. Drucker, B. Lee, P. Y. Simard, J. Suh. „ModelTracker: Redesigning Performance Analysis Tools for Machine Learning“. In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (2015) (zitiert auf S. 15).

[AHH+14] B. Alsallakh, A. Hanbury, H. Hauser, S. Miksch, A. Rauber. „Visual Methods for Analyzing Probabilistic Classification Data“. In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), S. 1703–1712 (zitiert auf S. 1, 15, 16, 22).

[Alp14] E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2014. ISBN: 0262028182 (zitiert auf S. 1, 3, 4, 7).

[Bis06] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738 (zitiert auf S. 3, 4, 6–9).

[Bos] M. Bostock. *Data-Driven Documents*. URL: https://d3js.org/ (zitiert auf S. 31).

[Dat] DataTables. *Add advanced interaction controls to your HTML tables the free easy way*. URL: https://datatables.net/ (zitiert auf S. 32).

[FAT+14] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A. Y. Zomaya, S. Foufou, A. Bouras. „A Survey of Clustering Algorithms for Big Data: Taxonomy and Empirical Analysis“. In: *IEEE Transactions on Emerging Topics in Computing* 2.3 (2014), S. 267–279 (zitiert auf S. 10).

[FCT11] G. Fedynyshyn, M. C. Chuah, G. Tan. „Detection and Classification of Different Botnet C&C Channels“. In: *Autonomic and Trusted Computing*. Hrsg. von J. M. A. Calero, L. T. Yang, F. G. Mármol, L. J. García Villalba, A. X. Li, Y. Wang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 228–242. ISBN: 978-3-642-23496-5 (zitiert auf S. 1).

[FHM09] C. Ferri, J. Hernandez-Orallo, R. Modroiu. „An Experimental Comparison of Performance Measures for Classification“. In: *Pattern Recognition Letters* 30 (Jan. 2009), S. 27–38. DOI: 10.1016/j.patrec.2008.08.010 (zitiert auf S. 1, 5, 6).

[FM11] I. Fette, A. Melnikov. *The WebSocket Protocol*. Dez. 2011. URL: https://www.hjp.at/doc/rfc/rfc6455.html (zitiert auf S. 31).

[GBYH20] M. Gleicher, A. Barve, X. Yu, F. Heimerl. „Boxer: Interactive Comparison of Classifier Results“. In: *Computer Graphics Forum* 39 (Juni 2020), S. 181–193. DOI: 10.1111/cgf.13972 (zitiert auf S. 1, 16, 17).

[Jas13] F. A. Jassim. „Image denoising using interquartile range filter with local averaging“. In: *arXiv preprint arXiv:1302.1007* (2013) (zitiert auf S. 12).

[KKEM10] D. Keim, J. Kohlhammer, G. Ellis, F. Mansmann, Hrsg. *Mastering the information age : solving problems with visual analytics*. Goslar : Eurographics Association, 2010. ISBN: 978-3-905673-77-7. URL: https://diglib.eg.org/handle/10.2312/14803 (zitiert auf S. 11, 12).

[LZY17] M. Li, L. Zhen, X. Yao. „How to read many-objective solution sets in parallel coordinates [educational forum]“. In: *IEEE Computational Intelligence Magazine* 12.4 (2017), S. 88–100 (zitiert auf S. 13).

[Mar] J. T. Mark Otto. *Bootstrap*. URL: https://getbootstrap.com/ (zitiert auf S. 31).

[Mur12] K. P. Murphy. *Machine Learning - A Probabilistic Perspective*. Cambridge: MIT Press, 2012. ISBN: 978-0-262-01802-9 (zitiert auf S. 4–8).

[Pot06] K. Potter. „Methods for presenting statistical information: The box plot“. In: (Jan. 2006) (zitiert auf S. 12, 13).

[RAL+17] D. Ren, S. Amershi, B. Lee, J. Suh, J. D. Williams. „Squares: Supporting Interactive Performance Analysis for Multiclass Classifiers“. In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2017), S. 61–70 (zitiert auf S. 1, 15, 19–21, 35, 41).

[RJ13] H. Reiterer, H.-C. Jetter. „Informationsvisualisierung“. In: *Grundlagen der praktischen Information und Dokumentation : Handbuch zur Einführung in die Informationswissenschaft und - praxis*. Hrsg. von R. Kuhlen, W. Semar, D. Strauch. 6. Aufl. Berlin [u.a.]: De Gruyter Saur, 2013, S. 192–206. ISBN: 978-3-11-025822-6 (zitiert auf S. 11).

[sci] scikit-learn. *Machine Learning in Python*. URL: https://scikit-learn.org/stable/index.html (zitiert auf S. 31).

[SZ16] A. K. Sood, S. Zeadally. „A Taxonomy of Domain-Generation Algorithms“. In: *IEEE Security Privacy* 14.4 (2016), S. 46–53 (zitiert auf S. 10, 11).

[WPB+19] J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. Viegas, J. Wilson. „The What-If Tool: Interactive Probing of Machine Learning Models“. In: *IEEE Transactions on Visualization and Computer Graphics* (2019), S. 1–1. ISSN: 2160-9306. DOI: 10.1109/tvcg.2019.2934619. URL: http://dx.doi.org/10.1109/TVCG.2019.2934619 (zitiert auf S. 16).

[XW09] R. Xu, D. Wunsch. *Clustering*. Wiley-IEEE Press, 2009. ISBN: 9780470276808 (zitiert auf S. 9, 10).

[ZWM+19] J. Zhang, Y. Wang, P. Molino, L. Li, D. S. Ebert. „Manifold: A Model-Agnostic Framework for Interpretation and Diagnosis of Machine Learning Models“. In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (Jan. 2019), S. 364–373. ISSN: 2160-9306. DOI: 10.1109/tvcg.2018.2864499. URL: http://dx.doi.org/10.1109/TVCG.2018.2864499 (zitiert auf S. 16, 17).

Alle URLs wurden zuletzt am 05. 11. 2020 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift