27. April 2020 Studie · Mobile Sicherheit · KI · 20 Min. Lesezeit

Ähnlichkeit von Android-Anwendungen über Deep Learning

Archiviert. Dieser Inhalt wird nicht mehr aktualisiert. Er bleibt zur Information verfügbar.

Stand 2020-04-24
Quelle https://technology2.a-sit.at/veroeffentlichungen/aehnlichkeit-von-android-anwendungen-ueber-deep-learning/

Angesichts der ungebrochenen Popularität von Anwendungen für die Mobilplattform Android finden sich in den „App Stores" zahlreiche Apps, die laut Beschreibung eine ähnliche oder identische Funktionalität implementieren. In vielen Fällen ist diese Beschreibung aber nicht existent, unvollständig oder schlichtweg falsch. Dieser Umstand ist insbesondere dann problematisch, wenn Anwendungen sensible Daten verarbeiten, auf kritische Sensoren zugreifen oder Daten von BenutzerInnen verarbeiten.

Im Zuge dieses Projekts sollen mithilfe von Autoencoder, einem Ansatz aus dem „Deep Learning"-Bereich, Ähnlichkeiten in den Beschreibungstexten, sowie den beanspruchten Berechtigungen von Android-Anwendungen, analysiert werden. Das Ziel ist es, besser zu verstehen, ob und inwiefern sich Applikationen anhand ihrer Beschreibungstexte ähneln und ob Android-Apps mit vergleichbarer Funktionalität auch die gleichen Berechtigungen benötigen.

1. Einleitung

Google Play und Plattformen von Drittanbietern stellen Android-BenutzerInnen eine breite Auswahl von Apps für diverse Zwecke bereit. Zur besseren Übersichtlichkeit werden Apps in der Regel in Kategorien gruppiert und ihr designierter Einsatzzweck in einem Satz zusammengefasst. Bei Auswahl einer gewissen Anwendung wird Interessierten eine ausführlichere Funktionsbeschreibung, Screenshots und eine Liste benötigter Berechtigungen angezeigt. Diese Metadaten stellen für BenutzerInnen die Entscheidungsgrundlage dar, ob eine Anwendung hinreichend vertrauenswürdig und funktional tauglich erscheint, um schließlich installiert zu werden.

Um den Ansprüchen von BenutzerInnen gerecht zu werden und das mögliche Verhalten von Apps vorhersehbar zu gestalten, schützt Android den Zugriff auf datenschutzkritische Gerätefunktionen. Wann immer eine App Zugriff auf die Kamera, das Mikrofon oder andere sensible Funktionen benötigt, werden

NutzerInnen um die entsprechende Erlaubnis gebeten. Mit Android 6 wurden die Berechtigungen neu organisiert, wobei sog. „gefährliche Berechtigungen“ [1] all jene subsumieren, die Zugriff auf heikle Gerätefunktionen und persönliche Daten von BenutzerInnen haben.

Aktuelle Studien [1] [2] zeigen, dass vielen BenutzerInnen das Risiko nicht bewusst ist, das mit der Gewährung gewisser Berechtigungen einhergeht. Obwohl es prinzipiell intuitiv erscheint, dass z.B. eine Messenger-App Zugriff auf die Kontakte benötigt, um herauszufinden, wer über die Anwendung kontaktiert werden kann, so kann mitunter nicht offensichtlich sein, warum die gleiche App auch die Berechtigung anfordert, den aktuellen Standort mittels GPS festzustellen. In der Praxis kommt erschwerend hinzu, dass auch Malware und tief in die Privatsphäre eingreifende Anwendungen mehr Berechtigungen anfordern, als für ihre vermeintliche Funktionalität eigentlich notwendig wäre. In den vergangenen Jahren hat Google als treibende Kraft hinter der Entwicklung von Android mehrere Anläufe unternommen, um die Beschreibung und Darstellung von Berechtigungsanfragen transparenter und einfacher verständlich zu gestalten. Nichtsdestotrotz gibt es zurzeit noch keine Lösung, die benötigte Berechtigungen hinsichtlich des Einsatzzwecks einer App darstellen würde.

Um versehentliche Leaks persönlicher Daten zu verhindern, ist es essentiell, dass BenutzerInnen verdeutlicht wird, in welchem Kontext Apps auf sensitive Daten, das eingebaute Mikrofon, die Kamera oder weitere Sensoren zugreifen. Eine wesentliche Quelle, um herauszufinden, welche Funktionalität eine App abbildet, ist die von Herstellern bereitgestellte Beschreibung. Im Idealfall ist sie so verfasst, dass sich nicht nur die Funktionalität einer App erschließt, sondern implizit oder explizit auch der Grund für die Berechtigungen, die eine Anwendung benötigt. In der Praxis finden BenutzerInnen jedoch oftmals Beschreibungen vor, die nur minimal oder unzutreffend sind, oder sicherheitsrelevante Angaben über ihren Umgang mit sensiblen Daten grundsätzlich aussparen. Die für dieses Projekt zentralen Fragestellungen lauten daher wie folgt:

1. Inwiefern drücken Beschreibungen von Android-Apps sicherheits- und datenschutzkritische Aspekte überhaupt aus? Lassen sich Korrelationen zwischen der tatsächlichen Implementierung und den von Herstellern gegebenen Beschreibung eruieren?

2. Beinhalten Anwendungen, die von ihrer Funktionsbeschreibung her ähnlich erscheinen, auch verwandte Implementierungen sicherheitskritischer Funktionen? Stehen die in der Beschreibung verwendeten Wörter, Wortgruppen und Phrasen in Zusammenhang mit dem Vorkommen konkreter, sicherheitsrelevanter Funktionen?

Moderne Android-Anwendungen bestehen mitunter aus tausenden von Klassen, die nicht alle in gleichem Ausmaß für die Sicherheit von Apps relevant sind. In der Praxis inkludieren Applikationen zudem häufig Bibliotheken von Drittanbietern, aus denen nur ein minimaler Funktionsbaustein benötigt wird. Der tatsächliche Zugriff und die Verwendung sicherheitskritischer Daten spiegelt sich jedoch vor allem im Aufruf von Systemschnittstellen [2] (APIs) wider. Per Definition abstrahieren sie unter anderem den Zugriff auf SMS, Kamera, GPS, sowie weitere Ressourcen. Die Kontrolle 1 https://developer.android.com/guide/topics/permissions/overview#dangerous_permissions 2 https://source.android.com/security/overview/app-security#the-android-permission-model-accessing-protected-apis darüber, ob und wie Apps diese Schnittstellen aufrufen dürfen, steuert das Berechtigungsmodell von Android. Bei der Installation und während der Ausführung angefragte Berechtigungen stehen daher *in direktem Zusammenhang* mit der Implementierung und dem Zugriff von

Apps auf sicherheitskritische Daten. In diesem Projekt nutzen wir diesen Umstand aus, betrachten App-Berechtigungen als Abstraktion von Aufrufen sicherheitskritischer Schnittstellen und setzen sie in Relation zu Herstellerbeschreibungen von Apps.

Eine Möglichkeit, um die Ähnlichkeit von Anwendungen zu bestimmen, ist, sie anhand gewisser Attribute in Gruppen bzw. „Cluster" zuzuordnen. Die Kernidee ist, dass Apps im selben Cluster über „ähnliche" Eigenschaften verfügen und sich von anderen Anwendungen, die nicht im selben Cluster sind, dadurch unterscheiden.

In diesem Projekt verfolgen wir das Ziel, einen „Deep Learning"-Ansatz zu entwickeln, der Android-Anwendungen anhand ihrer designierten Funktion in einem Cluster zuweist. Wir trainieren dazu zwei Autoencoder und wenden Algorithmen für Clustering und Dimensionalitätsreduktion auf ein Set realer Anwendungen an. Die Absicht ist es, die Nichtlinearität und adaptive Charakteristiken tiefer neuronaler Netze zu nutzen, um die Ähnlichkeit von Anwendungen schließlich einfach visualisieren zu können. Die Lösung soll zudem eingesetzt werden können, um „Outliers" zu identifizieren, d.h. beispielsweise Apps, die beispielsweise mehr Berechtigungen fordern als andere mit ähnlicher Funktionalität oder Apps, die sich funktional zwar ähneln, jedoch signifikant unterschiedliche Beschreibungstexte aufweisen.

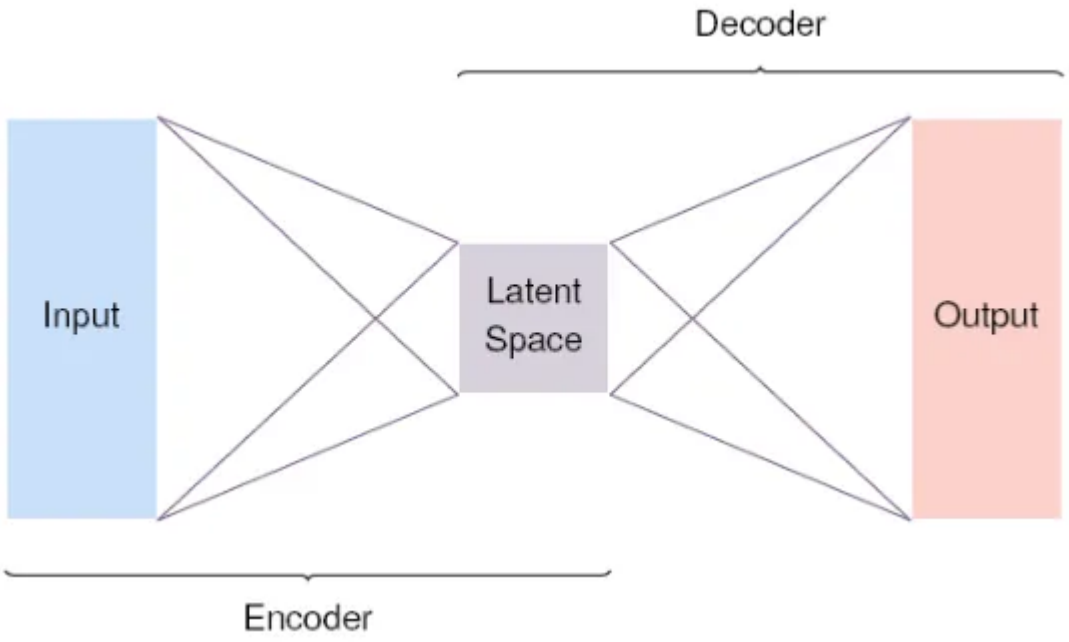
2. Neuronale Netze

In den vergangenen Jahren haben neuronale Netze wesentlich zur Lösung komplexer Probleme, etwa bei der Bildverarbeitung oder Spracherkennung, beigetragen. Die Verfügbarkeit und gute Skalierbarkeit von Rechenleistung haben wesentlich mitgewirkt, dass neuronale Netze heutzutage nur mehr oberflächlich an die zu verarbeitenden Daten angepasst werden müssen, selbstständig Muster (Patterns) erkennen und sie als wichtig oder unwichtig einordnen. Modelle, die Daten dabei sehr weit abstrahieren und größere Zusammenhänge erkennen, sind im Allgemeinen unter dem Begriff „Deep Learning" bekannt.

2.1. Autoencoder

Autoencoder gehören zur Gruppe von Algorithmen für „unsupervised learning". Im Gegensatz zum „supervised"-Ansatz werden die zu trainierenden Daten dabei nicht im Vorfeld mit einem „Label" versehen. Im konkreten Anwendungskontext heißt das, dass beim Training eines Autoencoders mit Android-Apps a priori keine Zuordnung vorgenommen werden muss, um dem Netzwerk mitzuteilen, welche Apps zueinander ähnlich sind.

Das Funktionsprinzip von Autoencoder, wie in *Abbildung 1* schematisch dargestellt, ist es, selbstständig eine mathematische Transformation zwischen Input und Output festzustellen. Dazu erhält ein Autoencoder während des Trainings die gleichen Daten als Input und Output. Um eine Abstraktion von gegeben Daten zu erreichen, befindet sich zwischen Input- und Output-Layer *Abbildung 1. Wesentliche Elemente eines Autoencoders*



Durch den Unterschied bei der Dimensionalität ist das neuronale Netzwerk gezwungen, essentielle Charakteristika und Unterschiede in den Trainingsdaten zu lernen, sie zu abstrahieren und in einem Layer niedrigerer Dimensionalität darzustellen. Diese Schicht wird auch „Latent Space“ genannt. Die Grundidee der Abstraktion ist es, nur unwesentliche Informationen wegzulassen und für jedes gegebene Sample immer noch genug Charakteristika zu kennen, um es eindeutig wiedererkennen zu können. Der „Latent Space“ kann daher auch als komprimierte Form von Daten gesehen werden.

Ein Autoencoder stellt Abhängigkeiten unter gegebenen Eingabedaten in einem nichtlinearen Raum dar. Die Vektorrepräsentation des „Latent Space“ kann folglich zur Dimensionalitätsreduktion und Clustering verwendet werden. Traditionelle Algorithmen für Clustering wie etwa k-means versuchen Ähnlichkeiten direkt auf Basis des Inputs festzustellen. Bei Daten mit vielen wesentlichen Attributen (Features) kann dies jedoch zur Darstellung von Abhängigkeiten in Dimensionen führen, die in einfachen Clustern nicht mehr dargestellt werden können. Autoencoder tragen diesem Problem Rechnung, indem gegebene Daten vor dem Clustering in einen Teilraum mit niedrigerer Dimension projiziert werden. Hierauf lassen sich schließlich konventionelle Algorithmen für Clustering anwenden.

2.2. Dimensionalitätsreduktion über t-SNE

t-SNE [3] ist ein Verfahren um Daten aus einem nichtlinearen, hochdimensionalen Raum in 2D oder 3D visualisieren zu können. Neben der Hauptkomponentenanalyse (PCA) ist t-SNE ein weit verbreiteter Ansatz zur Darstellung von Daten auf Basis von Wahrscheinlichkeitsschätzungen.

„Stochastic Neighborhood Embedding“ (SNE) berechnet Wahrscheinlichkeiten auf Basis von Punktzu-Punkt-Distanzen über den euklidischen Abstand. Die Grundidee ist, dass die „Nachbarschaft“ eines gewissen Datenpunktes mit einer Wahrscheinlichkeitsfunktion ausgedrückt werden kann. Naheliegende Punkte haben dabei eine hohe Likelihood, weit entfernt liegende eine nahe null.

In der Praxis gibt es beim Einsatz von t-SNE mehrere Aspekte, die beachtet werden müssen. Zum einen verwendet t-SNE Gradientenabstieg, d.h. wenn es mehrere lokale Minima gibt, kann es mehrere mögliche Lösungen geben. Darüber hinaus bietet t-SNE einen Hyperparameter „perplexity“, der je nach Daten bzw.

individuellem Fall konfiguriert werden kann. Die Kernidee besteht darin, die Wichtigkeit einzelner benachbart liegender Punkte in Relation zu weiter entfernt liegenden Punkten zu gewichten.

Für die „Perplexity“ werden typischerweise mehrere Konfigurationen visualisiert und schließlich jene gewählt, die augenscheinlich die sinnvollsten Ergebnisse produziert. Die in *Abbildung 2* exemplarisch *Abbildung 2. t-SNE-Daten visualisiert mit* gezeigte Visualisierung von Daten veranschaulicht die *unterschiedlicher „Perplexity“* Unterschiede in Abhängigkeit der Wahl des Hyperparameters. Mit einer „Perplexity“ von 10 sind im konkreten Beispiel etwa keine individuellen Datenpunkte erkennbar, da sie zu komprimiert ausgegeben werden. Eine „Perplexity“ von 45 wiederum zeigt eine große Punktewolke, die visuell keine klare Abgrenzung zwischen den Punkten trifft. Man könnte somit annehmen, dass sich die gegebenen Daten nicht sauber in Cluster unterteilen lassen. Eine „Perplexity“ von 25 veranschaulicht jedoch drei klar voneinander separierte Cluster, die den tatsächlichen Verhältnissen (im ersten Quadranten dargestellt) sehr nahekommen.

In diesem Projekt setzen wir t-SNE zur Visualisierung von Clustern ein. Um geeignete Darstellungen von Clustern zu erhalten, werden für jede Darstellung alle Perplexitätsstufen von 2 bis 50 durchprobiert. Diese Herangehensweise wird auch von den Autoren von t-SNE empfohlen.

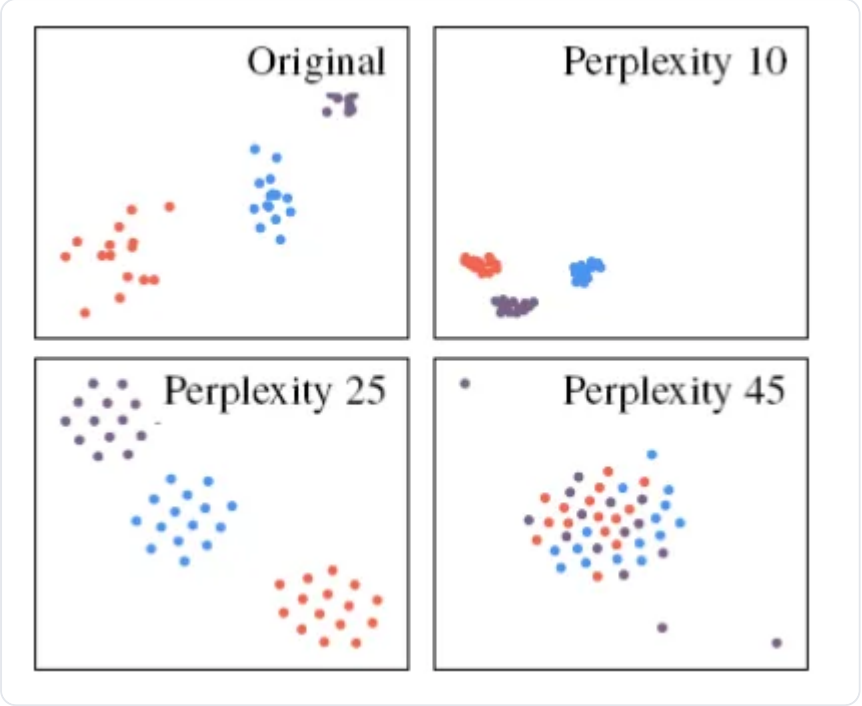


Abbildung 2. t-SNE-Daten visualisiert mit unterschiedlicher „Perplexity“

3. Lernen von Zusammenhängen

Um Android-Anwendungen auf Basis der angefragten Berechtigungen und Beschreibungstexte zu gruppieren, wird eine Kombination aus Autoencoder und t-SNE eingesetzt. Angepasst an die jeweiligen Inputdaten verwenden die Autoencoder unterschiedliche Architekturen und Datenstrukturen. Der „Latent Space“ wird mittels t-SNE visualisiert.

3.1. Verarbeitung von App-Berechtigungen

Der Autoencoder zur Repräsentation von Korrelationen in App-Berechtigungen (PAE) erhält als Input und Output jeweils einzelne Berechtigungen, ausgedrückt als Binärwerte. Wie in *Abbildung 3* dargestellt, besteht der erste Verarbeitungsschritt im Auslesen der Berechtigungen aus dem Manifest, das jeder Android-App beiliegt. Jeder der dort vorgefundenen Systemberechtigungen [3] wird ein eindeutiger Index zugewiesen, um anschließend eine Vektordarstellung zu ermöglichen. Im Vorverarbeitungsschritt stellen wir alle von einer App beanspruchten Berechtigungen in einem Zeilenvektor dar. Der Index eines jeden Elements repräsentiert eine unterschiedliche Berechtigung. Verwendet eine App beispielsweise die Berechtigung mit Index 50, so ist das 50. Element des Vektors auf 1 gesetzt, ansonsten auf 0. Als Resultat erhalten wir also für jede App einen sog. „onehot-encoded"-Vektor, der alle beanspruchten Berechtigungen binär darstellt.

Abbildung 3. Vorverarbeitungsschritte beim Autoencoder für App-Berechtigungen

3.2. Verarbeitung von App-Beschreibungen

Analog zu Berechtigungen trainieren wir einen zweiten Autoencoder, der als Input und Output die Beschreibungstexte von Android-Apps verwendet. Die einzelnen Wörter eines gegebenen Textes werden dabei als TF-IDF-Vektoren [4] gewichtet.

Ausgehend von einzelnen Wörtern eines Beschreibungstextes werden folgende Vorverarbeitungsschritte vorgenommen, um in weiterer Folge aussagekräftige TF-IDF-Vektoren daraus abzuleiten:

1. **HTML-Tags:** Beschreibungstexte enthalten oft HTML-Tags zur Formatierung. Diese Verweise werden vollständig entfernt, um nur mehr den reinen Text zu erhalten.

2. Entfernen von nicht-alpanumerischen Zeichen und „Stopwords": Semantisch irrelevante Zeichen, wie etwa Bindestriche, Apostrophe oder Interpunktionen werden nicht weiter betrachtet. Darüber hinaus werden Konjunktionen und sonstige Wörter entfernt, die so häufig vorkommen, dass sie nichts über individuelle Apps aussagen.

3. **Stammformreduktion („Stemming"):** TF-IDF würde Wörter wie „Datenbank" und „Datenbanken" als zwei separate Vektoren darstellen, die nicht notwendigerweise in semantischen Zusammenhang miteinander stehen. Um die verschiedenen morphologischen Varianten von Wörtern als zusammen gehörig darzustellen, wenden wir auf alle Nomen und Verben eine Stammformreduktion an.

Die resultierenden Tokens einer jeden App werden schließlich zu TF-IDF-Vektoren konvertiert und können in weiterer Folge als Input und Output für den Autoencoder verwendet werden.

3 https://developer.android.com/reference/android/Manifest.permission

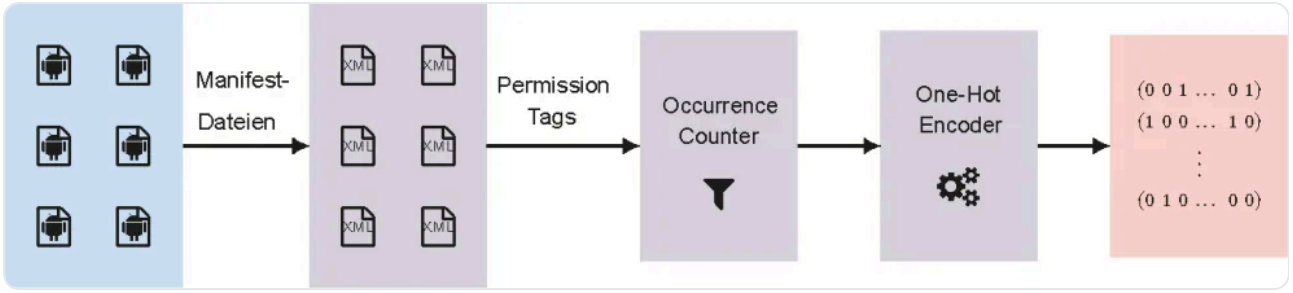


Abbildung 4: veranschaulicht die weitere Verarbeitung von Berechtigungen und Beschreibungstexten mithilfe von zwei Autoencoder. Da die Anzahl möglicher Berechtigungen beschränkt ist, können sie als „One-Hot"-Vektoren dargestellt werden. Die Wörter von Beschreibungen hingegen werden über TF-IDF entsprechend ihrer semantischen Relevanz gewichtet. Mit den jeweiligen Inputs und Outputs wird in weiterer Folge ein Autoencoder für Berechtigungen („Permission Autoencoder" – PAE) und ein weiterer für Beschreibungen („Description Autoencoder" – DAE) trainiert. Die Ausgaben des jeweiligen „Latent Space" können in weiterer Folge mithilfe von t-SNE 2 visualisiert werden.

3.3. Training der Autoencoder

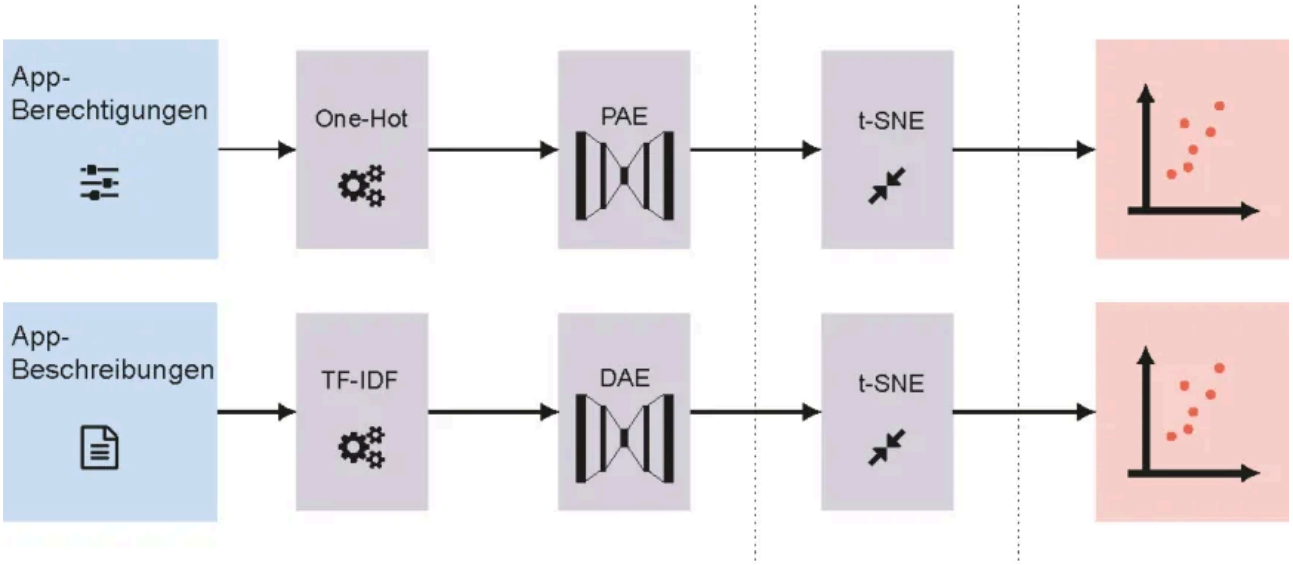
Abbildung 4. Verarbeitung von Berechtigungen und Beschreibungen über Autoencoder Nach dem Trainieren des Autoencoders beschreibt der „Latent Space" des PAE die von einer App verwendeten Berechtigungen in einem 5-dimensionalen Vektorraum. Analog dazu drückt der DAE Zusammenhänge in Beschreibungen über einen 10-dimensionalen Vektorraum aus. Diese Größen ergeben sich durch folgende Überlegungen, die der Architektur des Autoencoders zugrunde liegen:

- Um eine abstrakte bzw. komprimierte Darstellung von Trainingsdaten abzuleiten, muss mindestens ein Layer des Autoencoders eine geringere Dimension aufweisen, als die gegebenen Daten.
- Encoder und Decoder sind zueinander symmetrisch bzw. teilen die gleiche Architektur. Um die Komplexität des „Deep Learning"-Ansatzes nach oben hin zu limitieren, definieren wir zwischen 1 und 8 „hidden layer" als sinnvolle Zielgröße.
- Der Layer des „Latent Space" des PAE soll aus fünf „hidden neurons" bestehen, der des DAE aus 10. Diese Werte folgen keinen strikten Richtlinien, sondern sind empirisch gewählt. Grundsätzlich gilt, dass ein Layer umso weniger versteckte Neuronen haben soll, je mehr Schichten das Netzwerk insgesamt umfasst. Andernfalls würde ein Autoencoder seiner Aufgabe nur unzureichend nachkommen, Daten zu abstrahieren.

Weitere Hyperparameter, mit denen das Netzwerk trainiert wird, sind in Tabelle 1 zusammengefasst.

Optimizer	Adam
Batchgröße	32
Initialisierung	Uniform
Loss-Funktion	Binary Cross-Entropy (PAE) Mean-Squared Error (DAE)
Aktivierungsfunktion bei hidden layers	ReLU
Aktivierungsfunktion final	Linear
Early Stopping	10 Epochen

Tabelle 1. Gewählte Hyperparameter für die Autoencoder



4. Evaluierung

Die nachfolgende Evaluierung verfolgt in erster Linie das Ziel, die Performance der trainierten Autoencoder im Hinblick auf die Anwendung mit realen Android-Apps zu überprüfen. Mithilfe von t-SNE und PCA soll stichprobenartig untersucht werden, inwieweit sich auf Basis von Berechtigungen und Beschreibungstexten eine Ähnlichkeit von Apps feststellen lässt.

4.1. Datensatz

Um die Autoencoder mit den in Android-Apps enthaltenen Berechtigungen zu trainieren, werden Archive einer Anwendung, sowie auch zugehörige Beschreibungen benötigt. Da ein Crawlen des Google PlayStore technischen Beschränkungen unterworfen ist, verwenden wir für die nachfolgende Analyse die Sammlung des PlayDrone-Projekts [5].

Nach dem Download von rund 115.000 Apps [4] mitsamt Beschreibungen, werden zunächst Cross-Platform-Apps aussortiert, bei denen die Implementierung nicht als Dalvik Bytecode vorliegt, sondern in Form einer JavaScript-Webanwendung realisiert ist. Darüber hinaus wird für alle Samples sichergestellt,

dass die resultierenden TF-IDF-Vektoren für Input und Output jeweils Werte über null beinhalten. Dadurch lässt sich von vorneherein verhindern, dass Samples verwendet werden, die keinen erkennbaren Informationsgehalt aufweisen.

Für die zu trainierenden App-Berechtigungen und Beschreibungen werden außerdem folgende Einschränkungen getroffen:

- **App-Beschreibungen:** Der Text muss auf Englisch vorliegen und muss, nach Verarbeitung wie in Abschnitt 3.2 beschrieben, mindestens aus 30 Zeichen bestehen.
- **App-Berechtigungen:** Eine Anwendung soll zumindest eine Berechtigung anfordern.

4.2. Clustering

Nach dem Trainieren der beiden Autoencoder mit dem im vorigen Abschnitt beschriebenem Datensatz, können die resultierenden Modelle dafür verwendet werden, für beliebige weitere Apps anzugeben, mit welchen bekannten Anwendungen Ähnlichkeiten bestehen. Die in diesem Evaluierungsszenario verfolgten wissenschaftlichen Fragestellungen lauten wie folgt:

1. „Können anhand der Beschreibungstexte von Apps Gruppen bzw. Kategorien von Apps identifiziert werden?" Da die Zuweisung von Apps in Kategorien in Google Play und Plattformen von Drittanbietern zumeist nur vergleichsweise generisch vorgenommen wird (z.B. „Produktivität", „Werkzeuge", „Business", etc.), wäre es interessant, genauere Abstufungen zu erhalten. Autoencoder ermöglichen durch ihre Abstraktionsfähigkeit die gewünschte Darstellung ohne vorangehendes manuelles „Labelling" von App-Samples.

2. „Unterscheiden sich Apps mit unterschiedlichem Einsatzzweck auch beim Set der beanspruchten Berechtigungen? Wenn ja, benötigen Anwendungen mit ähnlicher Funktionalität auch ein ähnliches Set an Berechtigungen?" Um darzustellen, welche Zusammenhänge die Autoencoder konkret lernen konnten, werden die trainierten Modelle in weiterer Folge mit manuell ausgewählten Stichproben getestet. Primär soll dabei festgestellt werden, ob die gegebenen Apps in plausible Cluster eingeordnet werden und welche anderen Anwendungen ebenfalls im gleichen Cluster vorzufinden sind.

Die zur Visualisierung zwangsläufig vorzunehmende Dimensionalitätsreduktion kann unter anderem mittels Hauptkomponentenanalyse (PCA) oder t-SNE erreicht werden. Da PCA ein linearer Algorithmus ist, kann er im Gegensatz zu t-SNE nicht immer sinnvoll eingesetzt werden, um komplexe, nicht lineare Abhängigkeiten, wie im Fall vorliegend, abzubilden. Der auf Wahrscheinlichkeitsmodellen basierende t-SNE-Algorithmus ist wiederum auf die Wahl der richtigen 4 https://archive.org/details/playdrone-apks „Perplexity"-Stufe angewiesen. Da sich nicht pauschal und vor allem a priori sagen lässt, welcher Algorithmus brauchbarere Visualisierungen liefert, werden in den nachfolgend präsentierten Analysen die Ausgaben von PCA und t-SNE einander gegenübergestellt.

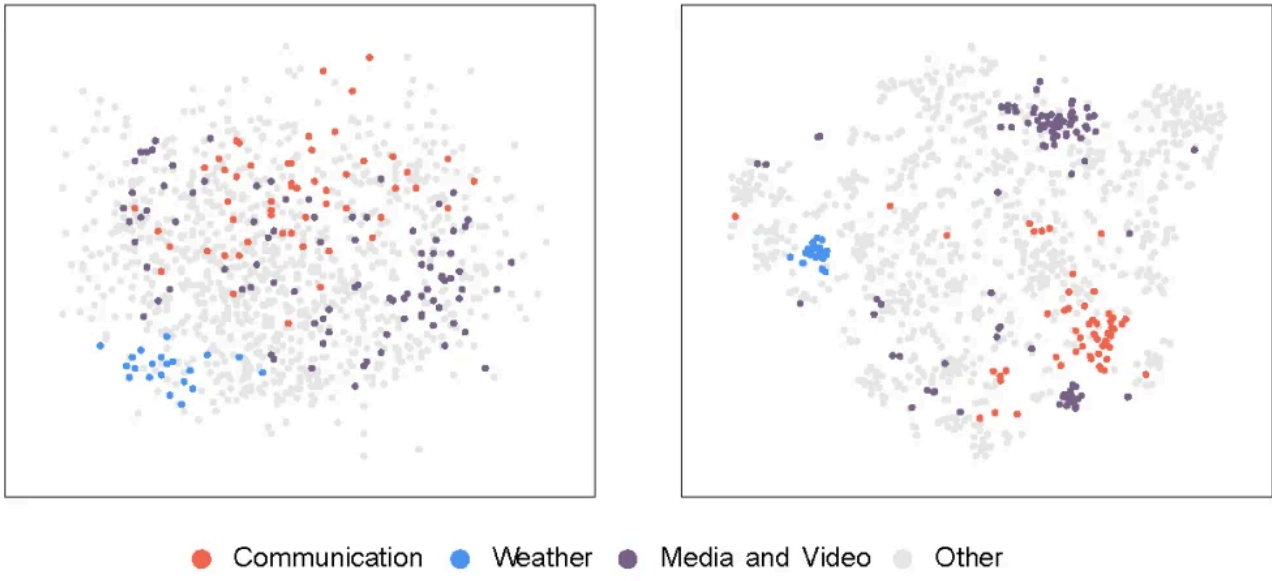
4.2.1. Ähnlichkeit von App-Beschreibungen

Für den im Weiteren angestrengten Vergleich nehmen wir an, dass textuell ähnlich beschriebene Apps der gleichen Kategorie zugewiesen werden sollten. Da der PlayDrone-Datensatz Anwendungen in 24 Kategorien aufbereitet, fokussieren wir uns aus Gründen der Übersicht auf drei ausgewählte: „Communication", „Weather" und „Media and Video".

PCA mit App-Beschreibungen t-SNE mit App-Beschreibungen *Abbildung 5. Cluster mit App-Beschreibungen, visualisiert über PCA und t-SNE* Nach dem Verarbeiten der Beschreibungstexte, Trainieren des Autoencoders und Anwenden von PCA und t-SNE veranschaulicht Abbildung 5 die resultierenden Cluster. Die färbig hervorgehobenen Punkte zeigen, dass t-SNE (mit „Perplexity 45“) Android-Apps einer gewissen Kategorie prinzipiell näher beisammen positioniert. Während bei PCA Apps der Kategorie „Weather“ vergleichsweise kompakt gruppiert sind, kommt diese Gruppierung bei t-SNE noch umso stärker zum Ausdruck. Auch für die anderen beiden Kategorien ist erkennbar, das mit t-SNE eine wesentlich eindeutigere Zuweisung zu in sich homogenen Cluster durchführbar ist.

WhatsApp Messenger	**Col2**	**Col3**
**# **	**PCA**	**t-SNE**
**1 **	Little Photo	Mercury Messenger (Free)
**2 **	Mobo Video Player Pro	Yahoo Messenger Plug-In
**3 **	Textgram – Instagram Text	Telegram
**4 **	Photo Warp	Yahoo Messenger
**5 **	Meme Generator Free	Azar-Video Chat&Call Messenger

Tabelle 2. Nachbar-Apps von “WhatsApp” im Cluster Zusätzlich zur visuellen Evaluierung der Beschreibungen, wählen wir in den Punktewolken vorkommenden Anwendungen aus und führen die benachbarten Punkte an. Durch Berechnung und Sortieren des euklidischen Abstands zwischen einer Ausgangs-App und allen umgebenden Anwendungen, lassen sich z.B. die fünf am nächsten liegenden Apps für den Messenger „WhatsApp“ eruieren. Wie in Tabelle 2 aufgelistet, liefern PCA und t-SNE dabei unterschiedliche Ergebnisse. Es fällt auf, dass PCA insbesondere Apps findet, die einen Multimedia-Bezug



aufweisen, wohingegen t-SNE die Messenger-bezogenen Eigenschaften der App deutlicher berücksichtigt. Analog dazu zeigt Tabelle 3 die Ergebnisse für die Browser-Apps „Opera Mobile“. Es ist erkennbar, dass PCA beim Browser beispielsweise mehr „Productivity“-bezogene Apps auflistet und t-SNE sich mehr auf weitere Browser konzentriert. In beiden Beispielen weist t-SNE als Nachbarn jeweils nur Applikationen des gleichen Typs aus (Messenger und Browser), PCA jedoch nicht. Daraus lässt sich schließen, dass t-SNE im konkreten Fall sinnvollere App-Cluster ableitet.

Opera Mobile Browser	**Col2**	**Col3**
**# **	**PCA**	**t-SNE**
**1 **	JuiceDefender – battery saver	Next Browser for Android
**2 **	SmartWho Task Manager	Opera Mini – Fast web browser
**3 **	Battery Widget	Boat Browser for Android
**4 **	Desktop VisualizeR	Opera Mobile Classic
**5 **	History Eraser – Privacy Clean	Maxthon Browser – Fast

Tabelle 3. Nachbar-Apps von “Opera Mobile Browser” im Cluster Die im vorigen Abschnitt angeführte Fragestellung, ob sich anhand der Beschreibungstexte Kategorien von Apps in einzelnen Cluster wiederfinden, kann unzweifelhaft bejaht werden. Die in Abbildung 5 gezeigten Punktewolken bestätigen somit die Hypothese, dass Apps, die einer gewissen Kategorie angehören, auch eine semantisch ähnliche Beschreibung aufweisen. Dieser Umstand ist nicht selbstverständlich, da App-Hersteller beim Upload von Apps zu Google Play die Kategorie selbst wählen und Beschreibungstexte selbst definieren können. Korrelationen zwischen einzelnen Apps dergleichen Kategorie bestätigen implizit also auch, dass Hersteller überwiegend eine Kategorie gewählt haben, die mit dem Beschreibungstext korreliert.

Bemerkenswert ist ebenso die semantische Nähe des Beschreibungstextes von „WhatsApp [5] “ mit dem von anderen Messenger wie „Telegram [6] “. Ein direkter Vergleich der Texte in Google Play zeigt intuitiv eine verwandte Funktionalität an. Dennoch sind die Texte so formuliert, dass es für einen Autoencoder nicht möglich wäre, nur auf Basis des Vorkommens einzelner Schlüsselwörter eine Korrelation der beiden Apps abzuleiten. Die dennoch überaus korrekte Gruppierung bezeugt somit die Abstraktionsfähigkeit von Autoencoder und die Tatsache, dass die als Input und Output verwendeten Schlüsselwörter über TF-IDF scheinbar richtig gewichtet wurden.

4.2.2. Unterschiede bei App-Berechtigungen

Im Hinblick auf die zweite in Abschnitt 4.2 angeführte Fragestellung wird der Fokus auf Unterschiede in der Verwendung von Berechtigungen gelegt. Davon ausgehend, dass Apps mit ähnlichem Verhalten ähnliche Berechtigungen beanspruchen, wäre es interessant, vor allem jene Apps zu identifizieren, deren Berechtigungen von der Norm abweichen. Als konkrete Stichprobe wird im Weiteren der Schwerpunkt auf Anti-Malware-Apps und Web-Browser gelegt. Zur Visualisierung kommt neuerlich t-SNE zum Einsatz.

Anti-Malware-Apps Abbildung 6 zeigt in der linken Hälfte die Cluster-Zuordnung 17 Anti-Malware-Apps anhand der von ihnen beanspruchten Berechtigungen. Neben einer größeren, grün markierten Ansammlung von Apps, ist erkennbar, dass „Antivirus Free“, „Antivirus for Android“ und „Dr. Web Lite“ deutlich außerhalb dieses Clusters liegen. Die Nachforschung in den App-Archiven, inwiefern sich die angeforderten Berechtigungen von weiteren „Anti-Malware-Apps“ unterscheiden, zeigt, dass diese drei Apps keine Berechtigungen benötigen, um auf den Standort, Anruflisten, SMS oder Speicher zuzugreifen. Das Gros der anderen, in der grün markierten Punktewolke vorzufindenden Apps, benötigt hingegen sehr wohl Zugriff auf diese persönlichen Ressourcen von BenutzerInnen und verlangt dafür die Gewährung von 33 bis 81 Berechtigungen. Die „Outliers“ hingegen benötigen nur 5 bis 14 Berechtigungen. Naturgemäß benötigen Apps zum Auffinden von Malware jedoch auch

5 https://play.google.com/store/apps/details?id=com.whatsapp&hl=en 6 https://play.google.com/store/apps/details?id=org.telegram.messenger&hl=en Zugriff auf kritische Ressourcen, um suspektes Verhalten zu analysieren. Es erscheint daher offensichtlich, dass sich der Funktionsumfang dieser Apps signifikant von anderen Apps mit dem gleichen designierten Einsatzzweck unterscheiden muss.

Anti-Malware-Apps Web-Browser *Abbildung 6. Cluster von App-Berechtigungen, visualisiert über PCA und t-SNE*

Web-Browser Das rechte Teilbild in Abbildung 6 veranschaulicht die Cluster-Zuordnung von 21 Web-Browsern. Der Plot zeigt für drei Browser „Opera Mini“, „Puffin“ und „Photon“ eine deutliche Distanz zu den anderen Clustern. Eine manuelle Analyse der Berechtigungen von „Opera Mini“ zeigt, dass diese Abweichung vor allem der Tatsache geschuldet ist, dass der Browser bewusst „schlank“ konzipiert ist und ohne weitere Berechtigungen auskommen möchte. Die anderen beiden Browser sind als „Outlier“ gekennzeichnet, da sie spezielle Berechtigungen benötigen, um auf gewissen Geräten im „Launcher“ eigene Icons unterzubringen. Sie unterscheiden sich damit von allen anderen Browsern.

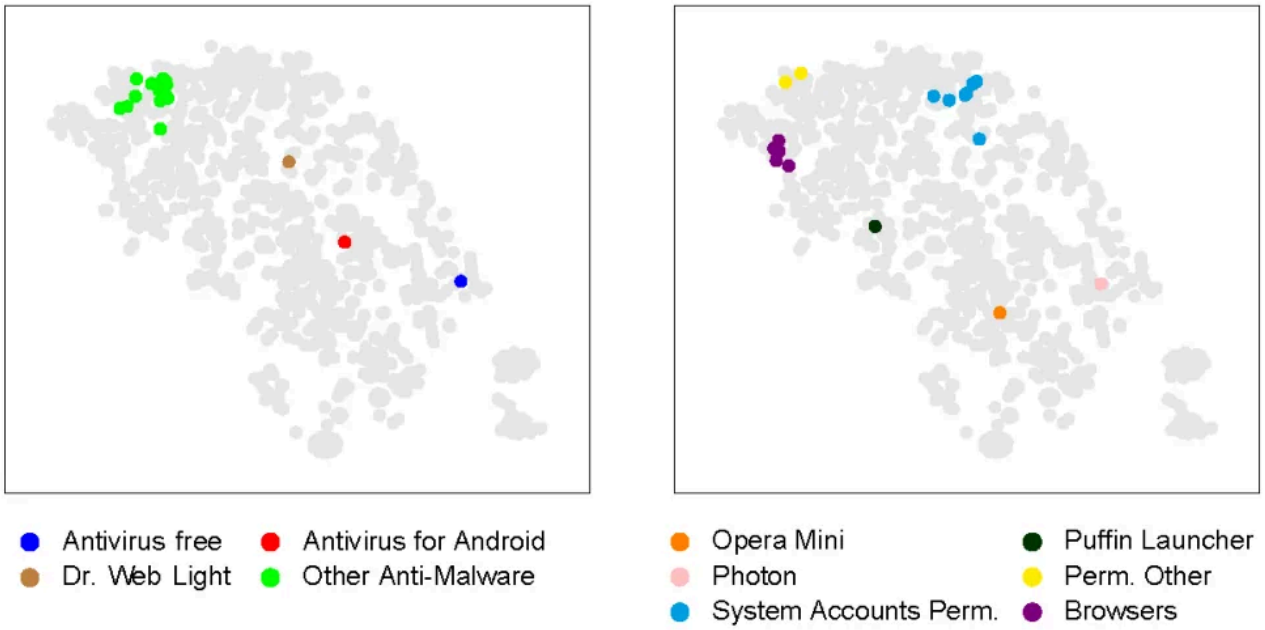
Die Abbildung zeigt, dass sich die meisten Browser in drei Kategorien aufteilen lassen. Der blau markierte Cluster beinhaltet insbesondere Apps, die sehr mit dem Betriebssystem verzahnt sind. Sie fordern beispielsweise Zugriff auf den eingerichteten Facebook- und Google-Account. Im Konkreten betrifft dies die Browser Firefox und Chrome. Apps in der lilafarbenen Gruppe benötigen zwar keinen Zugriff auf System-Accounts, fordern jedoch allerlei Berechtigungen ein, etwa um auf den Standort, das Mikrofon oder Systemeinstellungen zuzugreifen.

Obwohl sich Web-Browser nicht in einem großen Cluster wiederfinden, zeigen die einzelnen Teilcluster, dass die Kategorisierung nach Berechtigungen auch in diesem Fall gut funktioniert und auch feine Unterschiede bei Berechtigungen berücksichtigt. Empirisch erklärbar ist dieser Umstand vor allem in der Gewichtung durch TF-IDF, bei der häufig vorkommende (Kombinationen von) Berechtigungen als weniger relevant betrachtet werden, seltene (wie etwa die „Launcher“spezifischen) dafür umso mehr.

4.3. Fazit der Evaluierung

Basierend auf diesen Beispielen lässt sich schließen, dass der „Latent Space“ der Autoencoder mithilfe von t-SNE sowohl Ähnlichkeiten als auch Unterschiede sinnvoll visualisieren kann. Die exemplarische Auswahl von drei Kategorien zur Identifikation von Clustern bei App-Beschreibungen hat gezeigt, dass t-SNE einer

Darstellung mit PCA wesentlich überlegen ist. Auch ein individueller Vergleich einzelner Apps hat dieses Bild bestätigt, da die t-SNE gefundenen „Nachbarn“ funktional deutlich mehr miteinander verwandt sind.



5. Fazit

Im Zuge dieses Projekts wurde der Fokus auf eine vergleichende Analyse von Ähnlichkeiten in den Beschreibungstexten und beanspruchten System-Berechtigungen von Android-Apps gelegt. Das Ziel bestand darin, einen Ansatz zu entwickeln, um automatisiert feststellen zu können, welche Android-Applikationen funktional miteinander verwandt sind und inwiefern sich die Nähe zueinander in App-Beschreibungen und Berechtigungen widerspiegelt.

Die Vielzahl und der Umfang heutiger Android-Apps erschweren das Erkennen sinnvoller Zusammenhänge zwischen Apps. Um diese Abhängigkeiten dennoch analysieren zu können, wurde in diesem Projekt ein „Deep Learning“-Ansatz entwickelt, bei dem Autoencoder Informationen aus gegebenen App-Samples in abstrahierter bzw. komprimierter Form abbilden. Das Ergebnis lässt sich über Algorithmen wie PCA und t-SNE visualisieren und veranschaulicht Zusammenhänge in Form von Punktewolken und Clustern.

Der in diesem Projekt entwickelte Analyseansatz kann eingesetzt werden, um aufzuzeigen, welche Anwendungen von ihrer Beschreibung her ähnlich sind und daher mutmaßlich eine verwandte Funktionalität implementieren. Darüber hinaus eignet sich das Konzept, um Apps zu finden, die mehr, weniger, oder andere Berechtigungen fordern als Anwendungen mit einem vergleichbaren Einsatzzweck. Das Ergebnis unterstützt die sicherheitsorientierte Analyse von Mobilanwendungen und hilft zu verstehen, inwiefern sich Apps voneinander unterscheiden.

Literaturverzeichnis

[1] A. Peruma, J. Palmerino und D. Krutz, „Investigating user perception and comprehension of Android permission models,“ *MOBILESoft ‘18: Proceedings of the 5th International Conference on Mobile Software Engineering and Systems,* 2018.

[2] S. Ruberto und G. L. Scoccia, „An Investigation into Android Run-time Permissions from the End Users' Perspective," *MOBILESoft '18: Proceedings of the 5th International Conference on Mobile Software Engineering and Systems,* 2018.

[3] L. van der Maaten, „Visualizing Data using t-SNE," *Journal of Machine Learning Research, Volume 9,* 2008.

[4] T. Mikolov, I. Sutskever und K. Chen, „Distributed Representations of Words and Phrases and their Compositionality," *Neural Information Processing Systems NIPS,* 2013.

[5] N. Viennot, E. Garcia und J. Nieh, „A Measurement Study of Google Play," *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems,* 2014.

[6] S. Mingshen, M. Li und J. C. Lui, „DroidEagle: Seamless Detection of Visually Similar Android Apps," *8th ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec '15.,* 2015.

[7] Y. Shao und X. Luo, „Towards a scalable resource-driven approach for detecting repackaged Android applications," *Proceeding ACSAC '14 Proceedings of the 30th Annual Computer Security Applications Conference,* 2014.

[8] S. Lundberg und S.-I. Lee, „A Unified Approach to Interpreting Model Predictions," *Advances in Neural Information Processing Systems 30,* 2017.