Mini Use-Case 11: Logistische Regression für das klassifizieren von Bildern

Ein allzu menschliches Problem ist das Verlegen von Dingen. Begonnen bei simpelsten Objekten wie einer Fernbedienung ist das Suchen zeitaufwendig und mühsam. Umso mühsamer wird das Problem, wenn wir durch das Verlegen von Dingen Zeit und damit Geld verlieren. Ein Beispiel wäre das Verlegen von Werkzeug in einer Werkstatt. In diesem Use Case wollen wir prüfen ob auf einem Arbeitsplatz ein bestimmtes Werkzeug (ein Hammer) vorhanden ist oder nicht.

Aus technischer Sicht ist das Erkennen eines Hammers nicht trivial, da die Position und Orientierung sich immer ändern wird, die Beleuchtung ist nicht konstant oder das Werkzeug verändert sich durch die Nutzung optisch. Wir brauchen ein „Modell“, welches in der Lage ist, selbstständig einen Hammer zu identifizieren. Dies Implementierung des Modells nennt man auch Klassifizierungsmodell. Für den vorliegenden Use-Case werden wir eine logistische Regression verwenden, um ein Klassifizierungsmodell zu implementieren. Die logistische Regression ist eines der simpelsten Klassifizierungsmodell, welches mittels eines vorgegebenen Datensatzes eine lineare Klassifizierungsfunktion lernen kann und anschließend neue Bilder (hoffentlich richtig) klassifiziert. In unserem Fall wird der Datensatz zwei Klassen an Bildern enthalten, die dem Modell zeigen, wie ein Arbeitsplatz mit Hammer und wie ein leerer Arbeitsbereich aussieht. Wir erhoffen uns damit, dass wenn wir dem trainierten Modell anschließend ein neues Bild zeigen, dass klassifiziert werden kann zu welcher Klasse das Bild gehört. So versuchen wir Veränderungen der Orientierung bzw. Position– sprich wo sich das Werkzeug befindet und wie es da liegt – zu kompensieren.

Für Menschen ist die Klassifizierung von Bildern meist sehr einfach und läuft nahezu automatisch ab. Es ist oft einfach zu sagen ob auf einem Bild beispielsweise ein Hund, eine Katze oder ein Fahrrad zu sehen ist. Ein Computer verarbeitet Bilder anders und interpretiert diese als Matrizen. Vergleichbar ist dies mit einer großen Tabelle, wo in jeder Zeile und Spalte ein Zahlenwert steht. Jede Zahlt gibt eine Intensität eines Pixels an. Ein Pixel ist daher nichts anderes als ein Eintrag in der Tabelle. So kann beispielsweise ein Farbbild aus drei Tabellen bestehen. Die drei Tabellen oder eben Matrizen beinhalten die Intensität für Rot, Grün und Blau (RGB). „Legt“ man diese Matrizen nun übereinander erhält man ein Farbbild so wie man es typischerweise kennt. Für den Use-Case werden Bilder verwendet, welche 416x416 Pixel beinhalten. Sprich der Computer kennt zu jedem Bild drei Tabellen mit jeweils 416 Spalten und 416 Zeilen und der entsprechenden Farbintensität jedes Pixels.

Um nun für die Aufgabenstellung einen Weg zu finden die Bilder zu klassifizieren muss ein passendes Modell ausgewählt werden. Für diesen Use-Case wird, wie schon angesprochen, eine sogenannte logistische Regression ausgewählt. Um die Unterschiede zu anderen Algorithmen zu sehen sind die Use-Cases 11, 12, 13 und 15 mit derselben Aufgabenstellung ausgelegt und behandeln jeweils einen anderen Algorithmus.

Zunächst müssen allerdings die Bildmatrizen aufbereitet werden, dass diese auch von der logistischen Regression verarbeitet werden können. Der Input für die logistische Regression – welche weiter unten und im AIAV-Video „Klassische Neuronale Netze“ genauer erklärt wird – ist von dem Modell und den Daten abhängig. In dem vorliegenden Use-Case ist das Modell der logistischen Regression mittels scikit learn [3] implementiert und benötigt wie jedes ML Modell einen Vektor als Input. Der Unterschied zwischen Matrix und Vektor und wie diese Umwandlung geschieht ist in Abbildung 1 dargestellt. Das Bild (1) wird in die drei Farbebenen unterteilt (2). Anschließend wird von jeder Farbe (RGB) die Matrix (3) in einen Spaltenvektor

umgewandelt (4). Das bedeutet, wenn die Matrix eine Größe von $m \times n$ besitzt so wird diese zu einem Vektor der Größe $(m \times n) \times 1$. Dies passiert von Schritt (3) zu Schritt (4). Bei einem 416x416x3 großen Bild, wie es bei diesem Use-Case der Fall ist, bekommt man einen Vektor der Größe 519168. Dies ist unser neuer Datenpunkt beziehungsweise der Input für die logistische Regression.

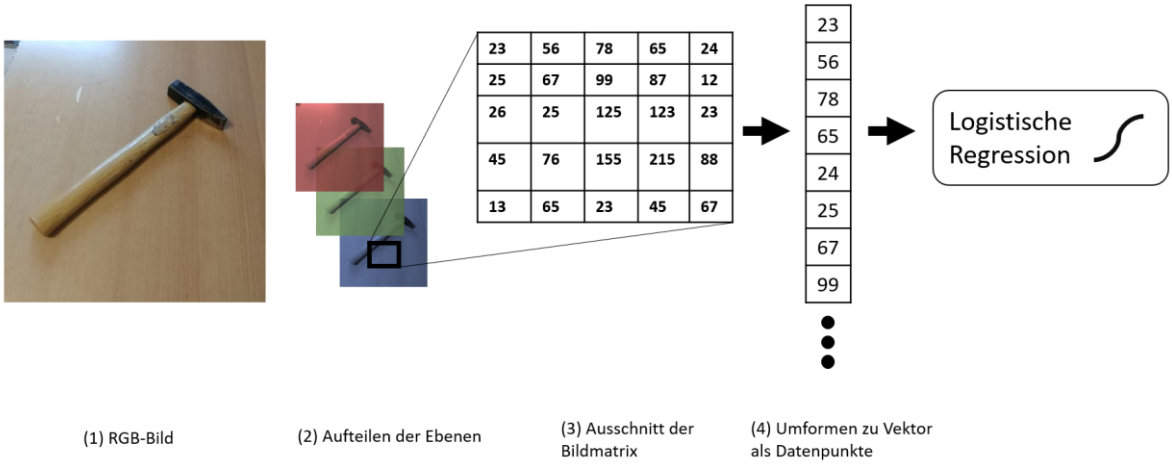

Abbildung 1: Aufbereiten der Daten (1) RGB-Bild welches in einen Spaltenvektor umgewandelt werden soll, (2) Aufteilung der Farbebenen in Rot, Grün und Blau, (3) Visualisierung eines Teiles der Bildmatrix, (4) Darstellung des Spaltenvektors im Vergleich zur Bildmatrix und Input für die logistische Regression)

Da das eine sehr hohe Menge an Zahlen ist werden zwei Schritte durchgeführt, um die Menge zu verringern. Als ersten Schritt wird das Originalbild verkleinert, da Details in diesem Fall nicht ausschlaggebend sind, sondern die gesamte Form des Werkzeuges, welches erkannt werden soll. Mit einem Viertel des Originalbildes werden die Datenpunkte auf eine Menge von 32448 reduziert. Anschließend wird der Datensatz mit einer Hauptkomponentenanalyse (auch PCA genannt) noch weiter verkleinert.

Die Hauptkomponentenanalyse erlaubt es uns, die Dimension von Datensätze (also die Länge vom Vektor) zu reduzieren. Dabei haben wir das Ziel die vermeidlich wichtigste Information zu behalten und potenziell irrelevante Information wegzulassen. Die Datenvektoren werden dabei in Komponenten zerlegt. Diese Komponenten beinhalten dann einen Teil der Information aus dem originalen Datensatz. Die Menge an verwendeten Komponenten kann dann so gewählt werden, dass ein bestimmter Anteil der Informationen aus dem originalen Datensatz erhalten bleibt. Das ermöglicht das Reduzieren auf nur 219 Datenpunkte. Das ist eine Reduktion von rund 99%. Weitere Informationen zur PCA findet man hier als Video oder als Use-Case.

In dem präsentierten Use-Case wird die Klassifizierung mittels PCA und logistischer Regression gelöst, um zu erkennen ob sich entweder ein Werkzeug im Arbeitsbereich befindet (Klasse 1) oder ob sich kein Werkzeug im Arbeitsbereich befindet (Klasse 2).

Eine logistische Regression nutzt eine sogenannte Sigmoidfunktion, welche Werte zwischen 0 und 1 annehmen kann und einer „S“-Form folgt. Eine solche Kurve ist in Abbildung 2 dargestellt. Wir hoffen, dass wir mit einer solchen Kurve die Bilder mit und ohne Hammer trennen können. Das Modell versucht beim Trainieren die Parameter zu finden, die dies bestmöglich erlauben. Stellen wir uns ein eindimensionales Problem vor – also die Reduktion der Bilder mittels PCA auf eine Zahl. Das ist in Abbildung 2 inklusive angepasster Sigmoidkurve dargestellt. Muss nun ein neues Bild – den das Model noch nie gesehen hat – klassifiziert werden, so kann einfach die PCA und anschließend die generierte Funktion verwendet werden. Der Algorithmus setzt hier das neue Beispiel ein und berechnet dann durch die angepasste Sigmoidfunktion den entsprechenden Wert zwischen 0 und 1. So ist beispielsweise ein Wert über 50% der Klasse 2 zuzuordnen und geringere Werte Klasse 1 [1],[2].

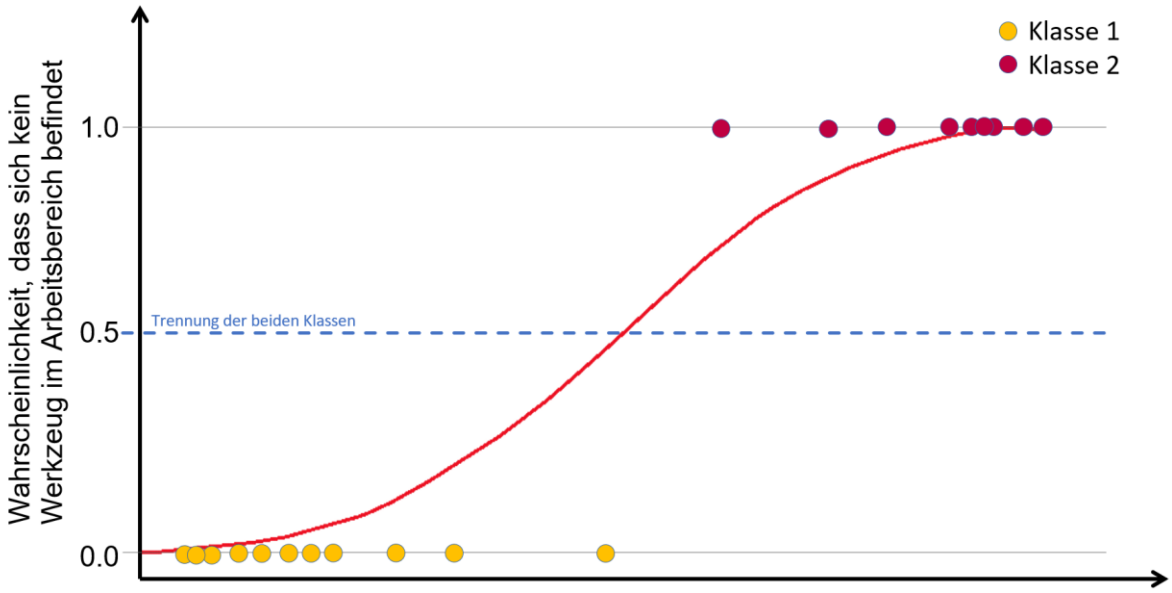

Abbildung 2: Darstellung der Daten und logistischen Regression

Wie hilft dies nun mit dem vorgestellten Use-Case?
In dem Use-Case geht es darum einen Hammer im Arbeitsraum zu erkennen. Sprich eine Klassifizierung von Bildern, ob es sich um Klasse 1 (Hammer im Bild) oder Klasse 2 (kein Hammer im Bild) handelt. Um eine Funktion zu finden, wird hier ein Modell eingesetzt, welches Vektoren (Datenpunkte), wie in Abbildung 2 vereinfacht dargestellt ist, einordnet. Wie schon besprochen können Bilder in Vektoren umgewandelt werden und somit für die logistische Regression eingesetzt werden. Über einen möglichst großen Datensatz kann dann eine Funktion gesucht werden, welche am besten zu den Trainings-Bildern passt. Wird nun ein (Live-)Video ausgewertet, kann nun jeder Frame (Bild-für-Bild) dem PCA und logistischen Regression Model gezeigt werden. Dieses wertet das Bild aus (klassifiziert es) wie es auch in der Theorie besprochen wurde. Anschließend bekommt man einen Wert, welcher eine prozentuelle Zuordnung ist zu welcher Klasse das Bild gehört. Wird das Bild der Klasse „Hammer“ zugeordnet, so sagt es aus, dass die KI einen Hammer im Arbeitsraum erkannt hat und dementsprechend ein Alarm oder Hinweis ausgegeben werden soll.

Das beiliegende Notebook/Code-Beispiel bezieht sich auf genauso einen Datensatz, welcher ebenfalls beigelegt ist. Fotos von einem Arbeitsplatz mit und ohne Hammer sind für das Trainieren abgelegt und werden von dem Algorithmus verwendet um anschließend eine Klassifizierung über den Webcam-Stream durchzuführen.

Wie schon angesprochen, ergeben die RGB-Bilder eine große Datenmenge. Um die Datenverarbeitung zu beschleunigen, wurden für diverse Tests die RGB-Bilder in Graustufenbilder umgewandelt, um aus den drei Farbmatrizen nur eine Matrix zu erstellen. Weiters wurde die PCA wie diskutiert eingesetzt.

Um die PCA zu trainieren wird die Implementierung der sklearn-Library [4] mit den Standardparametern verwenden. Der einzige Parameter welcher neu definiert wird ist die Anzahl der Komponenten. Für unser Beispiel werden 95% aller Features verwendet. Dieser Wert hat sich aus probieren mit dem gegebenen Datensatz ergeben.

Das Modell selbst wird trainiert, nachdem es definiert wurde. Hierzu bietet die sklearn-Library [3] die Möglichkeit aus verschiedenen Berechnungsmethoden und Einstellungen auszuwählen. Auf der sklearn-Webseite [3] ist genau beschrieben welcher solver sich für welchen Datensatz am besten anbietet. Trotz des recht kleinen Datensatzes wurde für den vorliegenden Use-Case der solver „saga“ ausgewählt. Dieser ist auch bei großen Datensätzen zu empfehlen.

Ist das Modell definiert wird mittels der Methode „fit“ trainiert. Hierzu wird der Datensatz zuerst geteilt. Insgesamt werden von den 2376 Bildern des gesamten Datensatzes 85% eingesetzt, um das Modell zu trainieren wobei die anderen 15% eingesetzt werden, um das Modell zu testen. In unserem Fall hat das trainierte Modell eine „Modell Score“ (eingebettete Implementierung zum Testen der Modelle von sklearn) von 0.96 aus 1.00 erhalten. Sollte dieser Wert nicht zufriedenstellend sein (zu niedrig), kann das an einem zu kleinem Datensatz oder zu simplen Modell (der logistischen Regression) liegen. Alternativen werden in den Use Cases 12,

13 und 15 diskutiert. Ebenso sollte das Problem des Over Fittings oder auch Überanpassung genannt, beachtet werden. Beim Over Fitting kann das Model nicht mehr zwischen relevanten Merkmalen entscheiden und lernt sozusagen den bekannten Datensatz auswendig. Tritt dies auf, kann das Model keine neuen Bilder klassifizieren, da es nicht die Merkmale (z.B. Kanten, Kreise, …) sondern die Bilder als ganzes gelernt hat. Um dies zu vermeiden können zum Beispiel die Parameter des Modells angepasst werden oder man überarbeitet den Trainings/Test-Datensatz.

Um ein besseres Ergebnis der Klassifizierung zu erhalten, empfiehlt sich, dass einerseits darauf geachtet wird, dass der Datensatz aussagekräftig und qualitativ hochwertig ist. Andererseits empfiehlt es sich auf ein komplexeres Modell umzusteigen, wie beispielsweise in den Use Cases 16 und 17 beschrieben. Diese behandeln die Themen CNN-Klassifizierer bzw. CNN-Detektor.

Ein solches Warnsystem wie es in diesem Use-Case beschrieben wurde, sorgt natürlich auch für Zweifel. Wird der oder die WerkstättenmitarbeiterIn überwacht? Jedes System kann für verschiedene Zwecke eingesetzt werden. Der Use-Case soll zeigen, dass AI nicht gefährlich oder einschränkend ist, sondern eben auch einem Menschen schützen und helfen kann. Die Zweifel sind allerdings berechtigt und sollten bei konkreter Implementierung solcher Systeme behandelt werden.

[1] Groß J. (2010) Logistische Regression. In: Grundlegende Statistik mit R. Vieweg+Teubner. https://doi-1org-10003429b00e7.han.technikum-wien.at/10.1007/978-3-8348-9677-3_22

[2] Schüler T. (2019) Logistische Regression. In: Kersting K., Lampert C., Rothkopf C. (eds) Wie Maschinen lernen. Springer, Wiesbaden. https://doi-1org-10003429b00e8.han.technikum-wien.at/10.1007/978-3-658-26763-6_14

[3] scikit-learn developers, „scikit-learn.org“, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. [Zugriff am 5 Oktober 2021].

[4] scikit-learn developers, „scikit-learn.org“,2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html [Zugriff am 21 Oktober 2021]