## Aufgaben

Dieser Workshop besteht aus vier Teilen, welche mit Anleitungen und gezielten Fragen einen Einstieg in die Welt des Machine Learning ermöglichen sollen. Mit 'optional' gekennzeichnete Fragen sind für tiefergehendes Verständnis gedacht, müssen aber nicht zwingend gelöst werden.

Bei den vorbereiteten Laptops befindet sich ein Ordner '`Workshop_Machine_Learning`' auf dem Desktop (Schreibtisch). Alle benötigten Dateien sind darin zu finden und Pfadangaben sind von diesem Ordner ausgehend zu verstehen.

Die Einführung, diese Anleitung sowie die verwendeten Datensets sind auf https://digitale-selbstbestimmung.info herunterladbar.

Diese Anleitung wurde für Weka Version 3.8.5 geschrieben. Mit anderen Versionen könnte das Interface oder die erwarteten Resultate sich ändern.

## Teil 1: Das Datenset

Start Weka mit einem Doppelklick auf '`Workshop_Machine_Learning/weka-8-3-5/weka.sh`'.

Klicke auf 'Explorer' in der rechten Leiste. Weka kann zu Beginn überwältigend sein und viele Einstellungen sind nicht ersichtlich. Wir empfehlen es ungeniert auszuprobieren, man kann nichts kaputt machen. Im schlimmsten Fall Weka einfach neu starten.

Lade das Datenset in '`Datensets/vote.arff`' in dem du auf 'Open-File' klickst und dich entsprechend durchmanövrierst.

Dieses Datenset kommt aus dem UCI Machine Learning Repository und beschreibt das Wahlverhalten amerikanischer Politiker:innen bei 16 wichtigen Abstimmungen nach Parteizugehörigkeit. Die offizielle Beschreibung liest sich so:

> *«This data set includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the Congressional Quarterly Almanac (CQA). The CQA lists nine different types of votes: voted for, paired for, and announced for (these three simplified to yea), voted against, paired against, and announced against (these three simplified to nay), voted present, voted present to avoid conflict of interest, and did not vote or otherwise make a position known (these three simplified to an unknown disposition).»*

Weitere Erklärung zu den Attributen im Datenset finden sich unter den folgenden Links. Diese werden nicht benötigt um die Aufgaben zu lösen, sie dienen nur dem Verständnis.

- https://archive.ics.uci.edu/ml/datasets/congressional+voting+records
- https://escholarship.org/uc/item/48r6d4z0

Das 'vote' Datenset besteht aus 17 Attributen, z.B. 'handicapped-infants', wobei wir das letzte Attribut ('Class') vorherzusagen versuchen werden. Jede Zeile Im Datenset ist ein

Parlamentarier. Das Attribut 'Class' beschreibt, ob es sich um einen 'Democrat' oder 'Republican' handelt. Alle anderen Attribute repräsentieren einzelne Stimmabgaben für die entsprechende Abstimmung und sind für jeden Parlamentarier entweder 'y' (ja), 'n' (nein) oder '' (leer, unbekannte Disposition).

Klicke in Weka auf 'Edit' um die Datenreihen anzusehen.

- *Frage 1.1: Wie viele Datenreihen gibt es?*
- *Frage 1.2: Was hat Datenreihe 4 bei 'physician-fee-freeze' geantwortet?*
- *Frage 1.3: Ist Datenreihe 4 Republikaner oder Demokrat?*

In der Weka-Bedienoberlfäche auf der rechten Seite sind die 17 Attribute aufgelistet. Die rote und blaue Farbe visualisiert den Anteil an Republikaner oder Demokraten, die jeweils 'y' (ja) oder 'n' (nein) gestimmt haben.

- *Frage 1.4: Wie viele 'Republicans' und wie viele 'Democrats' gibt es in dem Datenset?*
- *Frage 1.5: Kann man anhand des Attributes 'physician-fee-freeze' tendenziell sagen, ob jemand, der 'nein' gestimmt hat, Demokrat ist?*
- *Frage 1.6: Wie viele Abgeordnete haben bei 'crime' nein ('n') gestimmt?*

## Teil 2: Logistische Regression

Wir trainieren nun den Klassifikations-Algorithmus 'logistische Regression¨ auf dem 'vote' Datenset. Klicke dazu auf das Tab 'Classify' oben im Weka-Explorer. Wähle unter "Classifier" bei 'Choose' 'weka => classifier => functions => Logistic' aus.

Dieser Algorithmus sucht lineare Abhängigkeiten der Zielklasse aus den Attributen heraus und normalisiert die Resultate pro Klasse mit der logistischen Funktion, daher der Name. Für den Einstieg ist die genaue interne Funktionsweise des Algorithmus nicht relevant. Folgende Frage richtet sich daher an bereits erfahrenere Bearbeiter:innen und erfordert ein wenig Mathematikkenntnisse, Suchmaschinenhexerei und Wikipedialesen. Sie darf gerne übersprungen werden.

- *Frage 2.1: (optional) Wie berechnet die logistische Regression aus einer Datenreihe die wahrscheinlichste Klasse?*

Bei den 'Test options' gibt es nun mehrere Möglichkeiten, unser Datenset in Trainings- und Test-Daten zu unterteilen, welche nicht überlappen. Trainings-Daten werden verwendet, um die Parameter des Algorithmus zu trainieren. Trainieren bedeutet dabei, die trainierbaren Parameter (Zahlen) des Klassifikations-Algorithmus so zu setzen, dass die Zielklasse (Demokrat oder Republikaner) möglichst gut (auf den Trainings-Daten) vorhergesagt werden kann. Dieses Setzen erledigt der Klassifikations-Algorithmusspezifische Trainings-Algorithmus für uns.

Mit den Test-Daten können wir den zuvor trainierten Klassifikations-Algorithmus testen, wie gut er klassifiziert. Diese Test-Daten wurden beim Training nicht verwendet, die Parameter des Klassifikations-Algorithmus wurden also nicht darauf angepasst. Er hat diese Datenreihen also quasi 'nie gesehen'. Ihre Klassifizierung hat keinen Einfluss auf die Parameter und kann beliebig oft mit dem selben Ergebnis wiederholt werden.

Wir können einstellen, wie unsere Datenreihen in Trainings- und Testset unterteilt werden sollen. Für dem Moment stellen wir den Anteil an Trainings-Daten einfach auf 80% des gesamten Datensets. Wähle dazu den Punkt 'percentage split' aus und setze ihn auf '80'.

Wir wollen das Atributt 'Class' als Klassifizierungsziel nehmen. Falls nicht bereits eingestellt, wähle 'Class' mit dem Drop-Down-Menue über dem Knopf 'start' aus.

Nun können wir den Algorithmus mit Drücken auf 'start' trainieren. Trainiere ihn!

Im Fenster 'results list' erscheint nun für jedes Drücken auf 'start' ein Eintrag mit den Detailinformationen zum Training und Testen. Aufgrund unserer getätigten Einstellungen werden diese Details für die von uns gewählte Konfiguration sich jedoch nicht verändern. Im Fenster Rechts daneben werden die Details zum Algorithmus des ausgewählten Durchgangs angezeigt.

Scrolle im Detailfenster ('classifier output') ganz nach unten.

- *Frage 2.2: Was ist die Test-Genauigkeit unseres Tests? Wie viele Test-Datenreihen des Testsets wurden richtig klassifiziert. Wie viele falsch?*
- *Frage 2.3: Wie viele Test-Datenreihen wurden verwendet?*
- *Frage 2.4: Was verändert sich, wenn wir den 'percentage split' – den Anteil der Trainingsdaten – auf 90% erhöhen?*
- *Frage 2.5: Wie hat sich die Anzahl der korrekt klassifizierten Instanzen verändert?*
- *Frage 2.6: Wie hängt die Zahl 'total number of instances' mit der 'percentage split' zusammen? Kannst du einen konkreten Rechenweg finden?*
- Frage 2.7: Was wäre die Test-Genauigkeit (Prozentzahl), wenn wir stattdessen einfach eine Münze werfen würden?

Um bessere Statistiken zu erhalten, können wir den Algorithmus auch mehrmals trainieren, jeweils immer auf einem anderen Teil des Datensets. Die einfachste Methode ist 'cross-validation.' Zum Beispiel wird bei einer 5-fachen Cross-Validation das Datenset in 5 Teile geteilt. Dann wird der Algorithmus 5 mal trainiert, jeweils immer mit einem anderen der 5 Teile des Datensets als Test-Set, während die anderen 4 Teile zum Trainingset zusammengefasst werden. Die Anzahl der korrekt klassifizierten Test-Datenreihen wir dann am Ende als Summe über alle fünf Durchgänge ausgegeben.

- *Frage 2.8: Wird durch Cross-Validation die Fehler-Statistik genauer? Falls ja, warum?*

Führe ein Training mit 8-facher Cross-Validation aus. Wähle dazu 'Cross-validation' in den Test-Optionen aus und setze die Anzahl auf 8. Was beobachtest du?

- *Frage 2.9: Was ist die Anzahl an totalen Test-Datenreihen über die 8 Durchgänge? wie kommt diese Anzahl zustande?*

Damit sind wir am Ende dieses Teils angelangt. Zum weiteren Verständnis lohnt es sich auch noch folgende Fragen zu beantworten:

- *Frage 2.10: Ist es möglich, auf anderen Attributen als 'Class' zu trainieren? Falls ja, wie? Was bedeuten die Resultate?*
- *Frage 2.11: Wie hoch ist die Test-Genauigkeit, wenn wir nach 'immigration' klassifizieren. Warum? (Tipp: gucke dir die Verteilung von 'immigration' im Preprocess-Tab an)*

## Teil 3: Entscheidungsbäume (Decision Trees)

In diesem Teil analysieren wir ein Datenset, welches aufgrund der Symptome und weiteren Indikatoren das COVID-19-Testresultat vorhersagen soll. Aufgrund der strikten Datenschutz-Gesetze der Schweiz haben wir leider keinen Zugriff auf echte Daten, daher haben wir zum Zweck dieser Veranstaltung selbst welche generiert. Die Datenreihen haben keinen direkten Bezug zur Realität!

Das Datenset hat folgende Attribute, welche kurz vor den ausgedachten COVID-19-Tests aufgenommen wurde:

- days_since_dec_19: Anzahl Tage seit dem 1. Dezember 2019
- days_with_symptoms: Anzahl Tage mit Symptomen des Patienten vor dem Test-Tag, Ganzzahl.
- body_temp: Körpertemperatur des Patienten zum Testzeitpunkt, Kommazahl.
- swetting: Ob der Patient übermässig geschwitzt hat zum Testzeitpunkt. 'y' für ja, 'n' für nein.
- symptom: Stärkstes gezeigte Symptom des Patienten zum Testzeitpunkt. Ist entweder 'fever', 'head_ache', 'stomach_ache', 'nose_bleed', 'coughing', oder '' (leer).
- tested_positive: Ob der Test positiv war. 'y' für ja, 'n' für nein.

Lade das Datenset '`Datasets/covid-artificially-generated.csv`' in Weka. Eventuell musst du beim Öffnen im Suchdialog unten beim Dateityp von '`*.arff`' auf '`*.csv`' wechseln.

- *Frage 3.1: Was ist etwa der Anteil der positiv getesteten Personen? Ist das gross anders als beim vorherigen 'vote' Datenset?*
- *Frage 3.2: Gibt es Attribute, bei denen wir bei gewissen Werten wir klar 'tested_po-*

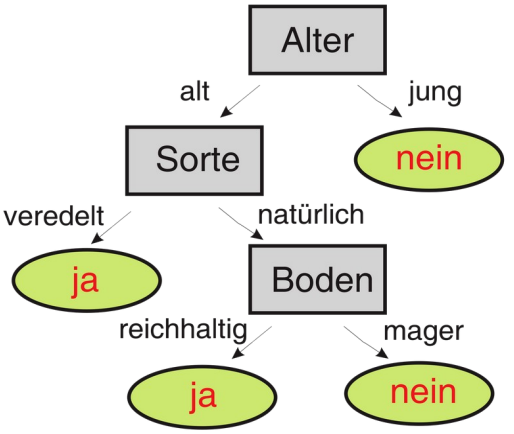
*sitive' = 'y' oder klar 'tested_positive' = 'n' klassifizieren können?*

- *Frage 3.3: Ist etwas problematisch an dieser Eindeutigkeit? Was wenn zukünftig Fälle auftauchen, welche nach den gegenwärtigen Daten klar als 'n‘ klassifiziert werden, aber eigentlich 'y‘ sind?*

Wechsle nun in das 'Classify'-Tab. Wähle beim 'Classifier' 'weka => classifier => trees => J48' aus. Dies ist ein sogenannter Entscheidungsbaum ('DecisionTree'), optimiert für Klassifikationen. Dieser Algorithmus ist bekannt als C4.5 und galt bis vor zehn Jahren als der beliebteste Klassifikationsalgorithmus, bevor er von komplexen neuronalen Netzen abgelöst wurde. Der Klassifikationsvorgang dieses Algorithmus wird gut durch Weka illustriert, daher erklären wir Entscheidungsbäume kurz.

Dabei nehmen wir ein einfacheres Beispiel. Hier seht ihr einen Entscheidungsbaum zur Vorhersage, ob ein Apfelbaum Früchte tragen wird:

Dabei sind Alter, Sorte und Boden Attribute, die in dem Fall nur zwei Kategorien ('jung'/'alt', 'veredelt'/'natürlich','reichhaltig'/'mager') beinhalten. Im unserem Beispiel der COVID-19-Daten haben wir auch normale Zahlen, für die der J48 Algorithmus dann einen optimalen Kipppunkt sucht. Das 'ja' und 'nein' bezieht sich auf das Zielattribut, ob der Baum Früchte tragen wird. Dabei wird jedes Ende der Entscheidung (also ein 'ja' oder 'nein' Knoten) als Blatt (Englisch 'leaf') des Entscheidungsbaumes bezeichnet.



*André Flöter via Wikipedia*

Wir trainieren jetzt unser COVID-Datenset. Stelle eine '10-fold Cross-Validation' ein, lasse das Zielattribute 'tested_positive' sein und trainiere den 'J48'-Algorithmus mit einem Klick auf 'start'.

Sieh dir nun die Details im Ausgabefenster an.

- *Frage 3.4: Kannst du herausfinden, wie der Algorithmus klassifiziert / wie er zu seiner Entscheidung ob 'tested_positive' gleich 'n' oder 'y' kommt? Erscheint dir dieser Algorithmus sinnvoll? Tipp: Es steht unter 'classifier model'.*

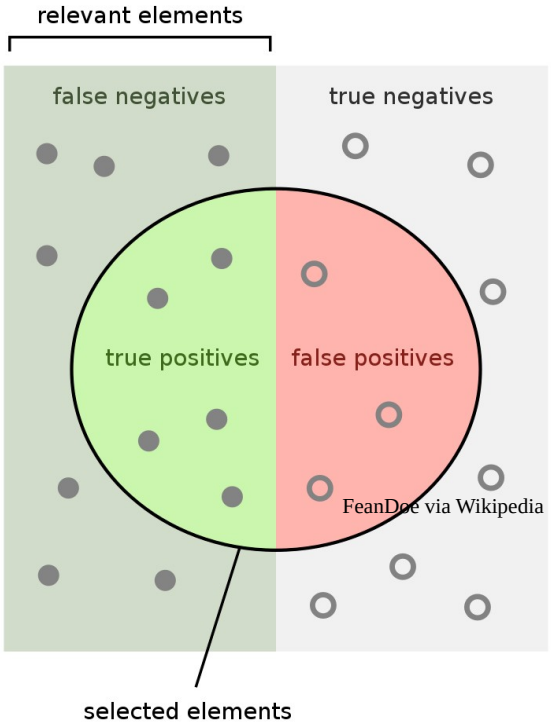
Jetzt begutachten wir die Genauigkeit des Algorithmus.

- *Frage 3.5: Was ist die durchschnittliche Genauigkeit des Algorithmus? Ist sie hoch?*

Der Algorithmus scheint die Test-Daten gut zu klassifizieren. Jetzt schauen wir aber noch die Details nach Klasse an. Diese sind in den Details unter 'Detailed Accuracy by Class' zu finden und unter 'Confusion Matrix' nochmals zusammengefasst. Hier sind einige Metriken gelistet, von denen wir zwei ausführen: 'True Positive' (TP) Rate und 'False Positive' (FP) Rate. Die weiteren Metriken wie 'Precision' und 'Recall' sind zwar wichtig, wenn man sich näher mit dem Thema befasst, aber sie sind nur die TP und FP Raten verschiedener Klas-

sen anders verrechnet. Der Informationsgehalt von TP und FP ist mit ihnen identisch.

- True Positive (TP) Rate: Der Anteil der Test-Datenreihen von der entsprechenden Klasse, welche auch als die entsprechende Klasse korrekt klassifiziert wurden.
- False Positive (FP) Rate: Der Anteil der anderen Klasse, die fälschlicherweise als diese Klasse klassifiziert wurde. Bei mehr als zwei Ziel-Klassen wird dabei der Schnitt des Wertes über alle anderen Klassen genommen.



FeanDoe via Wikipedia

Die Metriken TP rate und FP rate sind jeweils auf das Klassifizierungsziel-Attribut bezogen, die die Datenreihen haben, und nicht, als was sie vom Algorithmus klassifiziert wurden. Dies wird auch die Grund-Wahrheit (English: ground truth) genannt.

Wir kommen nun zu den letzten Fragen dieses Teils, welche ein fundamentales Problem offenbaren werden.

- *Frage 3.6: Wie viele Datenreihen der Klasse 'tested_positive' == 'n' wurden korrekt klassifiziert. Was ist ihr Anteil an allen Test-Datenreihen, welche als 'tested_positive' == 'n' klassifiziert wurden?*
- *Frage 3.7: Wie viele Datenreihen der Klasse 'tested_positive' == 'y' wurden korrekt klassifiziert. Was ist ihr Anteil an allen Test-Datenreihen, welche als 'tested_positive' == 'y' klassifiziert wurden? Siehst du etwas ungewöhnliches / anderes als im Falle 'tested_positive' == 'n' ? Was bedeutet das?*

Nun solltest du nur weiter lesen, wenn du die vorherige Frage beantwortet hast, weil wir jetzt einen Teil der Antwort erläutern.

Wie du vermutlich festgestellt hast, ist die Vorhersage-Genauigkeit für die Klasse 'tested_positive' = 'y' deutlich geringer als der über alle Test-Datenreihen berechneter Durchschnitt. Die Ursache liegt in der substanziell geringeren Anzahl von Datenreihe der Klasse 'tested_positive' = 'y', welche zwar im Einzellfall einen grossen Klassifizierungs-Fehler aufweisen, im Schnitt jedoch nicht viel zum Fehler beitragen weil ihr Vorkommen zu gering ist und der Fehler den Schnitt über alle Datenreihen berechnet. Der grosse Fehler wird erst ersichtlich, wenn man die seltene Klasse isoliert betrachtet. Wenn zum Beispiel ein Algorithmus wie bei uns 97% korrekt ist, dann können die übrigen 3% Fehler einen grossen Einfluss auf die Genauigkeit von Klassen haben, deren Häufigkeit sich im Bereich von 3% oder weniger bewegt.

Dieser Effekt ist im Englischen bekannt als Baserate-Fallacy, im Deutschen als Prävalenz-

fehler. Oft wird im öffentlichen Diskurs bei der Relevanz von bestimmten Algorithmen nur über die Genauigkeit gesprochen. Damit ist aber die Durchschnittsgenauigkeit gemeint und wie wir gesehen haben, greift diese in bestimmten Fällen zu kurz. Der Effekt verstärkt sich, je seltener gewisse Klassen werden, also je kleiner die Prävalenz (Häufigkeit, English baserate) einer Klasse ist.

Für solche Fälle produzieren derartige Algorithmen viele falsche Alarme. Für gewisse Anwendungen kann dies wünschenswert sein: Bei COVID-Testergebnissen hat man lieber grosse Fehler, als dass Ansteckungsherde entstehen. Aber in anderen Fällen jedoch kann es einen kritischen Einfluss auf Unschuldige haben, z.B. bei der automatisierten Detektion bei Strafverfolgungsbehörden und der darauf folgenden Sanktionierung von seltenen aber politisch hochgespielten Verbrechen. Je nach Seltenheit der Verbrechen geraten deutlich mehr Menschen ins Fadenkreuz der Ermittlungen, als tatsächlich schuldig wären.

Wenn ihr heute etwas mitnehmt, dann die Gefahr die Durchschnitts-Genauigkeit eines Algorithmus als Garantie für sein Funktionieren zu nehmen. Das darf man nicht und es wird trotzdem sehr oft so argumentiert. Um die Wirksamkeit eines Klassifikationsalgorithmus abzuschätzen benötigt man die 'True Positive' und 'False Positive Rate' (oder vergleichbare Metriken wie 'precision' und 'recall') von jeder Klasse. Die Gesamt-Genauigkeit alleine reicht nicht.

## Teil 4: Neuronales Netzwerk (Multi-Layer-Perceptron)

Dieser Teil richtet sich an schnelle Löser:innen und macht etwas grössere Sprünge als die Beispiele vorher.

Erfahrungsgemäss kann Weka auf schwächeren Computern Speicherprobleme bekommen, da es offenbar die Details bereits trainierter Algorithmen im Speicher behält und aufgrund dieses Speichermangels das Training neuronaler Netze nicht abschliessen kann. Daher empfiehlt es sich, jetzt Weka neu zu starten, oder dies spätestens dann zu tun, wenn die hier vorgeschlagenen Netze mehr als 5 Minuten Trainingszeit überschreiten.

Wir trainieren jetzt ein einfaches neuronales Netzwerk. Diese sind in der Literatur bekannt als Multi-Layer-Perceptron, aber auch als 'fully connected neural network'. Zeit für ein wenig Internet Recherche:

- *Frage 4.1: Wie funktionieren Multi-Layer-Perceptrons im Allgemeinen?*

Das Datenset, welches wir verwenden, findest du unter '`Datensets/letter.arff`'. Die offizielle Beschreibung:

*«The objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15. [..]»*

Weitere Infos zu den einzelnen Attributen findest du hier: https://archive.ics.uci.edu/ml/datasets/Letter+Recognition

Nachdem das Datenset geladen wurde, musst du unter 'Classify' den Algorithmus 'weka => classifier => functions => MultiLayerPerceptron' auswählen. Klicke danach auf den Text (Namen des Algorithmus) neben dem Knopf 'Choose'. Ein Fenster mit zusätzlichen Einstellungsmöglichkeiten öffnet sich. Stelle 'GUI' auf 'True' und 'Training time' auf 50.

- *Frage 4.2: (Optional) Was ist die 'training time'? (Vielleicht hat dir deine Internetrecherche was dazu verraten)*

Stelle ausserdem die Test-Evaluation auf 'percentage split' 90%. Das Training neuronaler Netze ist aufwändig und wir wollen es nur 1 mal durchführen, nicht 10 mal Cross-Validieren.

Wenn du jetzt auf 'start' klickst, öffnet sich ein Fenster, welches das neuronale Netz illustriert. Die Berechnung geht von den Attributen (den Features) links nach rechts zu den finalen Klassen. Jede schwarze Linie ist eine Zahl, ein Gewicht. Und jeder Punkt ist ein Node, wo die Inputs entsprechend den einzelnen Gewichten zusammengerechnet werden. Wie du siehst, hat bereits ein derart simples neuronales Netz sehr viele Parameter. Für fortgeschrittene Anwendungen wie Sprachsynthese rechnen heutige Modelle mit mehreren Milliarden Parameter (schwarzen Linien).

Man kann mit dieser GUI einiges anstellen, aber der Einfachheit halber schliessen wir diese wieder und deaktivieren sie auch in den Einstellungen. Klicke dazu auf den Algorithmusnamen neben dem Knopf 'Choose' und setze 'GUI' auf 'False'.

Wen du jetzt auf 'start' klickst, wird das neuronale Netz trainiert.

- *Frage 4.3: Wie hoch ist die Gesamt-Genauigkeit? Ist diese deutlich höher als wenn wir die Klasse einfach raten würden?*
- *Frage 4.4: Im Output-Fenster hat es ziemlich viele Zahlen im Abschnitt 'Classifier model (full training set)'. Was bedeuten diese genau?*

Damit ist dieser Workshop abgeschlossen. Wenn du magst, kannst du gerne die GUI wieder aktivieren und entdecken, was man damit machen kann. Du kannst auch weitere Parameter des neuronalen Netzwerkes ändern und dir in der GUI angucken, was der Effekt ist. Sei einfach gewarnt, dass die Rechenkomplexität schnell steigt und du unter Umständen beim Training geduldig sein musst.

Du darfst auch gerne andere Datensets, Algorithmen oder Einstellungen ausprobieren.

# Weiterführende Informationen

## Woher bekomme ich Datensets?

- UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets.php

- Kaggle: https://www.kaggle.com/datasets
- Suchmaschine deines geringsten Misstrauens
- Und viele mehr