h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbmn
FACHBEREICH MATHEMATIK
UND NATURWISSENSCHAFTEN

Gruppe 14
Tamara Döß (767956)
Tobias Rohrer (???)

Data Mining 1

Booklet von Gruppe 14

Inhaltsverzeichnis

Zusammenfassung

1 Aufgabe 1: Entscheidungsbäume

1.1 Feature Engineering

1.2 Analyse der Zielvariable

Wie Abbildung 1 entnommen werden kann ist Ausprägung der Zielvariable ungleich auf beide Klassen verteilt. Die Klassifizierungsgenauigkeit eines Modells muss demnach unter Berücksichtigung der sogenannten *Null Accuracy* bewertet werden. Unter *Null Accuracy* versteht man die Genauigkeit eines Modells, dass unabhängig von allen Eingaben immer die am häufigsten auftretende Klasse vorhersagt. In unserem Fall würde ein Modell, welches immer Regen vorhersagt, eine Klassifizierungsgenauigkeit von 79,39% erreichen. Das Ziel der nachfolgenden Schritte ist also ein Modell mit einer besseren Klassifizierungsgenauigkeit als die *Null Accuracy* aufzubauen.

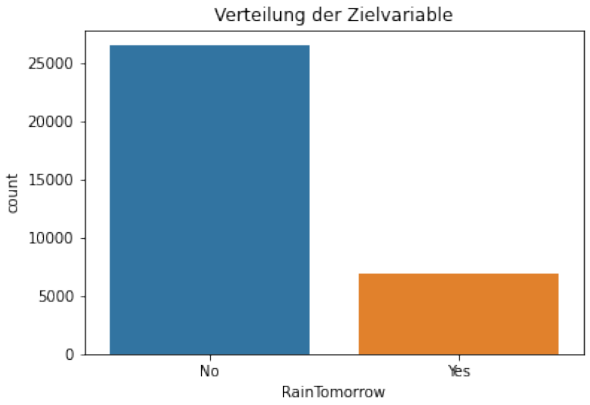

Abbildung 1: Verteilung der Zielvariable

1.2.1 Fehlende Werte

Der zu untersuchende Datensatz beinhaltet fehlende Werte. Diese müssen für eine weitere Verarbeitung bereinigt werden. Im Folgenden werden Methoden beschrieben, wie mit den fehlenden Werte umgegangen wurde:

Fehlende Zielvariable
Im ersten Schritt wurden 834 Beobachtungen, welche keinen Wert für die Zielvariable *RainTomorrow* aufweisen, aus dem Datensatz entfernt. Damit wurde die Anzahl an Beobachtungen auf 33402 reduziert.

Spalten mit fehlenden Werten
In einem nächsten Schritt werden die Spalten aus dem Datensatz entfernt, in denen mehr als 40% der beinhaltenden Variablen fehlen. Namentlich wurden somit die Spalten *Evaporation, Sunshine, Cloud9am* sowie *Cloud3pm* aus dem Datensatz entfernt. Der Schwellwert von 40% wurde empirisch festgelegt und hat zu den besten Klassifizierungsergebnissen geführt.

Beobachtungen mit fehlenden Werten
Des Weiteren werden Beobachtungen aus dem Datensatz entfernt, von denen mehr als 50%

der Variablen fehlen. Durch diesem Schritt wurden 55 Beobachtungen aus dem Datensatz entfernt.

Imputation

Durch die zuvor beschriebenen Methoden ist der Datensatz immer noch nicht frei von fehlenden Werten. Um diese zu ersetzen, werden für kategorische und numerische Variablen verschiedene Strategien zur Imputation verfolgt. Fehlende numerische Werte werden mit dem Median der jeweiligen Variable ersetzt. Der Median wurde gewählt, da dieser im Vergleich zum Mittelwert robuster gegenüber Ausreißern ist. Für kategorielle Variablen hingegen wird der am häufigsten vorkommende Wert verwendet. Wichtig bei der Ermittlung des Medians bzw. des häufigsten Wertes ist, dass dieser ausschließlich mit Hilfe der Trainingsdaten ermittelt wird. Es muss davon ausgegangen werden, dass die Testdaten nicht bekannt sind. Die Ermittlung auf Basis des gesamten Datensatzes, inklusive der Testdaten, würde zu *Data-Leakage* führen und ist zu vermeiden. Die auf Basis der Trainingsdaten ermittelten Werte für die Imputation werden auf die Trainings- und Testdaten angewendet.

1.2.2 Merkmalsaufteilung

Das Feld *Datum* wurde in die Merkmale *Year, Month* und *Day* aufgeteilt.

1.2.3 Merkmalserstellung

Eine weit verbreitete Technik des Feature Engineerings ist die Erstellung zusätzlicher Merkmalen. Somit wurde die Variable *MinMaxDiff* erstellt, welche die Differenz zwischen der minimalen und der maximalen Tages-Temperatur angibt. Des Weiteren wurden die Variablen *PressureDiff, HumidityDiff und WindSpeedDiff* als Differenz der Beobachtungen am Morgen und Abend erstellt.

1.2.4 Diskretisierung

Die Diskretisierung eines Merkmals kann eine Überanpassung bei der Erstellung von Modellen verhindern, indem der Wertebereich des Merkmals minimiert und somit generalisiert wird. Hierbei muss beachtet werden, dass der Informationsverlust durch die Diskretisierung nicht zu groß ist. Eine Diskretisierung wurde für das Merkmal *Month* durchgeführt, indem es in das Merkmal *Season* umgewandelt wurde. Das Merkmal *Season* fasst immer 3 Monate zu einer Jahreszeit zusammen.

1.2.5 Kodierung kategorischer Werte

Um kategorische Werte für weitere Analysen verwenden zu können, müssen diese in numerische Werte umkodiert werden. Hierbei wurden die folgenden Strategien Angewendet:

Binäre Kodierung

Die Zielvariable *RainTomorrow*, sowie die Variable *RainToday* liegen mit den binären Ausprägungen *Yes* und *No* vor. Für eine weitere Verarbeitung wurden die Ausprägungen in eine numerische Darstellung umgewandelt.

One-Hot-Kodierung

Das neu diskretisierte Merkmal *Season* wird mittels *One-Hot-Kodierung* umgewandelt. Eine einfache Kodierung mit einem Zahlenwertes pro auftretender Variablenausprägung

hat den Nachteil, dass dadurch eine Variable entsteht, die gegebenenfalls metrisch interpretiert wird.

Ziel-Kodierung
Mit Hilfe der Ziel-Kodierung werden die Merkmale *Location, WindGustDir, WindDir9am* und *WindDir3pm* umgewandelt. Hierbei werden die Merkmalsausprägungen als ihren Einfluss auf die Zielvariable kodiert.

1.2.6 Bereinigung von Ausreißern

Ausreißer können die Performance eines Modells mindern, indem sie als Hebelwerte agieren und somit die Schätzungen der Zielvariable verzerren. Aus diesem Grund werden die Merkmale des Datensatz hinsichtlich ihrer Ausreißer begutachtet. Es wird ersichtlich, dass nur die Merkmale *Rainfall* und *WindGustSpeed* abweichende Werte aufweisen. Das Entfernen dieser Werte aus dem Datensatz führt jedoch zu einer schlechteren Performance der im folgenden Abschnitt besprochenen Entscheidungsbäume. Deshalb werden die Beobachtungen nicht aus dem Datensatz entfernt. Zudem zählen die beiden Merkmale zu denjenige Merkmalen, die durch die Variablenselektion aussortiert und somit für weitere Modelle nicht mehr betrachtet werden.

1.2.7 Variablenselektion

Nach Abschluss der oben aufgeführten Schritte verfügt der Datensatz über 28 Einflussvariablen. Durch eine Variablenselektion soll die Anzahl dieser Einflussvariablen verringert werden. Das hat zum einen den Vorteil, dass Modelle besser interpretierbar sind. Zum Anderen wird die Generalisierungsfähigkeit eines Modells erhöht, indem Merkmale ohne, oder nur mit geringem Einfluss auf die Zielvariable entfernt werden. Die Gefahr einer Überanpassung wird somit minimiert

Für den betrachteten Datensatz wird eine univariate Variablenselektion mit dem Modul *SelectKBest* der Bibliothek *sklearn* durchgeführt. Dabei werden die k Merkmale ausgewählt, die den höchsten Wert der F-Statistik aufweisen. Dieser Wert gibt an, ob ein einzelnes Merkmal einen signifikanten Einfluss auf die Zielvariable hat. k wurde empirisch auf 5 festgelegt. Als Ergebnis wurden die Merkmale *WindGustDir, Humidity9am, Humidity3pm, RainToday* und *MinMaxDiff* ausgewählt.

1.3 Entscheidungsbäume

1.3.1 Aufteilung in Trainings- und Testdaten

Um das Modell nach Abschluss anhand der Klassifizierungsgenauigkeit bewerten zu können, sollte der Datensatz in Trainings- und Testdaten aufgeteilt werden. Die Aufteilung und eine anschließende Bewertung anhand der Testdaten ermöglicht eine Einschätzung der Generalisierungsfähigkeit des Modells. Als Aufteilungsverhältnis wurde 20% Testdaten und 80% Trainingsdaten gewählt. 20% der Daten entsprechen 6670 Datensätzen und bilden eine ausreichend große Menge um die Modellgüte zu bestimmen. Die Wahl des Aufteilungsverhältnisses wurde außerdem nach den Empfehlungen aus Geeron (2017) gewählt.

1.3.2 Standard Einstellungen

Quelle: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
Die Entscheidungsbäume werden mit dem Modul *tree.DecistionTreeClassifier* erstellt. Im ersten Aufgabenteil werden dazu die Default-Einstellungen des Moduls genutzt.

Diese geben an, dass ein Baum immer maximal gebaut wird (*max_ depth = None*). Das heißt, so lange mehr als ein Objekt in einem Knoten vorliegt, wird dieser Knoten weiter aufgespalten (*min_ samples_ split = 2*). Das Ergebnis ist überangepasst, da jeder einzelne Datenpunkt ein eigenes Blatt im Baum bekommt (*min_ samples_ leaf = 1*). Dieser Baum kann somit nicht auf unbekannte Daten generalisiert werden und der Testfehler fällt sehr hoch aus. Die Splits werden mit dem Gini-Wert durchgeführt und nicht mit der Entropie (*citerion=„gini“*). Zudem werden für jedes Spalten alle Merkmale einbezogen, um den besten *Split* zu finden (*max_ features = None*). In Abbildung 2 wird der mit den Default-Einstellungen erzeugte Baum dargestellt. Es ist leicht zu erkennen, dass dieser Baum zu viele Knoten und Blätter enthält

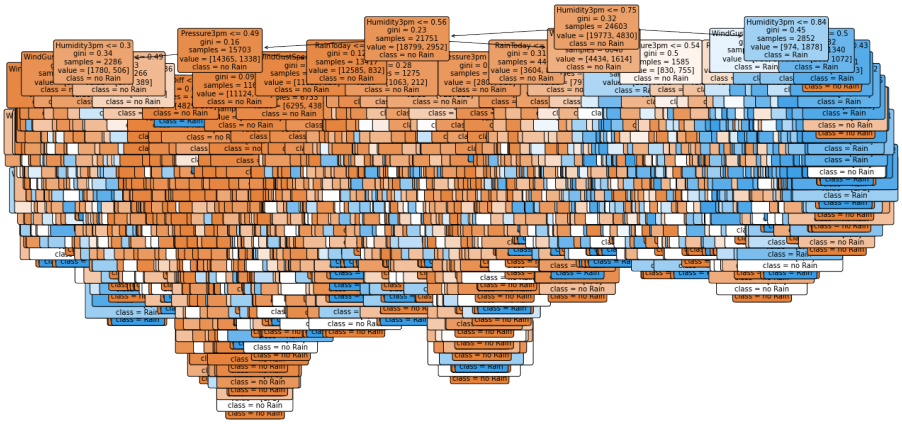

Abbildung 2: Entscheidungsbaum mit Default-Einstellungen

und somit eine Überanpassung darstellt. Diese Überanpassung kann auch an dem Trainings- und Testfehler abgelesen werden. Der Trainingsfehler beträgt für den gebildeten Entscheidungsbaum 0.0005 und der Testfehler 0.2340. Der Testfehler ist damit höher als bei einem Modell, das zufällig entscheidet. Dem kann mit verschiedenen Einstellungen entgegengewirkt werden.

1.3.3 Variationen

Um einen übersichtlichen und brauchbaren Entscheidungsbam erzeugen zu können werden verschiedenen Einstellungen für die Parameter *max_ depth, min_ ipurity_ decrease* und *criterion* angewandt. Im Folgenden werden die Entscheidungsbäume dargestellt, deren Parameter mit Hilfe des **GridSearch Verfahrens** festgelegt wurden. Bei der Visualisierung liegt der Fokus auf der Struktur des Baumes und nicht darauf, dass die einzelnen Knoten identifiziert werden können.

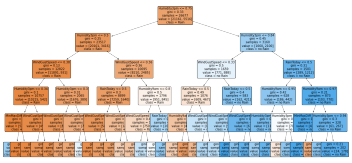

Abbildung 3: Maximale Tiefe von 5

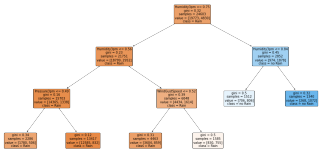

Abbildung 4: Minimale Unschärfe Reduktion von 0.003

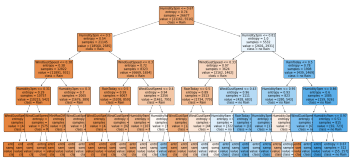

Abbildung 5: Entropie als Entscheidungskriterium

Maximale Tiefe

In Abbildung 3 wurde eine maximale Tiefe von **fünf** angegeben. Trotz der geringen Tiefe entstehen schon zu viele Blatt Knoten für eine übersichtliche Visualisierung. Im Gegensatz zu den anderen Variationen ist dieser Baum **symmetrisch**, da jeder Ast bis zur gleichen Tiefe verfolgt wird. Dieser Baum entspricht den ersten fünf Stufen des vorher erstellten Default-Entscheidungsbaum, da der *max_ depth*-Parameter den Baum einfach an der gegebenen Stufe abschneidet.

Unschärfe-Reduktion

In Abbildung 4 wurde mit dem Parameter *min_ impurity_ decrease* festgelegt, wie viel ein weiterer Knoten die **Unschärfe** des Entscheidungsbaumes reduzieren muss, um ausgebildet zu werden. Dabei wird jeder zu bildende Knoten einzeln bewertet, sodass es zu einem nicht symmetrischen Baum kommen kann, da nicht jeder Ast bis in die gleiche Tiefe verfolgt wird.

Entscheidungskriterium

In Abbildungen 5 wurde zusätzlich zur maximalen Tiefe von fünf das Entscheidungskriterium angepasst. Hier wird mit Hilfe der Entropie statt des Gini-Wertes entschieden, welcher Split durchgeführt wird. Die Struktur des Baumes wird dadurch nicht geändert, jedoch können die Knoten voneinander abweichen. Die Entropie erzeugt bei sonst gleich bleibenden Parametern einen Entscheidungsbaum, der einen kleineren Testfehler aufweist.

1.3.4 Minimal Cost-Complexity-Pruning

Im vorherigen Abschnitt wird mit Hilfe des *max_ depth* Parameters eine Überanpassung verhindert, es wird ein *pre-pruning* durchgeführt. Eine weitere Möglichkeit der Vermeidung des Overfittings bietet das *Cost-Complexitiy Pruning* als *post-pruning* Methode. Dabei wird der voll ausgebildete Baum iterativ beschnitten, indem diejenigen Teilbäume entfernt werden, die einen festgelegten penalisierten Fehlerterm minimieren.

Das Modul *DecisionTreeClassifier* arbeitet mit effektiven α-Werten der einzelnen Knoten. Für jeden Knoten des Baumes wird dieser Wert bestimmt. Dabei entpricht α_{eff} dem α, für das gilt: $R_\alpha(T_t) = R_\alpha(t)$. Der Knoten mit dem geringsten effektiven α wird vom Baum abgeschnitten. Dieses Vorgehen wird solange wiederholt, bis der geringste effektive α-Wert größer als der gegebenen Penalisierungs-Term *ccp_ alpha* ist.

Dabei ist zu beachten, dass die Bäume kleiner werden, also stärker beschnitten werden, je höher der gegebene Penalisierungs-Term ist, da dieser eine hohe Anzahl an Knoten bestraft und somit zu kleineren Bäumen führt (siehe auch A.1).

Das Pruning hat Auswirkungen auf die Modellgüte, indem es die Anpassung an die Trainingsdaten reguliert. Wird hohe Komplexität stark bestraft, kann der Entscheidungsbaum weniger gut an die Trainingsdaten angepasst werden. Wird der Penalisierungs-Term zu hoch gesetzt,

besteht die Gefahr der Unteranpassung, ein sehr niedriger Term führt zu Overfitting. in Abbildung 6 ist beispielhaft für einen Entscheidungsbaum mit Default-Einstellungen der Trainings- und Testfehler für verschiedene Pruning-Parameter α dargestellt.

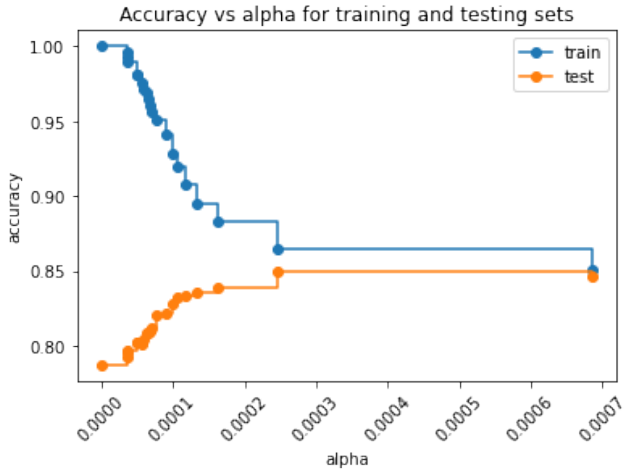

Abbildung 6: Trainings- und Testfehler für verschiedene α-Werte

2 Aufgabe 2: Neuronale Netze

2.1 Neuronale Netze mit Numpy

2.1.1 Implementierung Backprop mit einem Hidden Layer

2.1.2 Hyperparameter

Im folgenden werden Hyperparameter beschrieben, sowie ihre Auswirkungen auf das Neuronale Netzwerk diskutiert.

Anzahl der Neuronen im Hidden Layer

Wenn keine l_1 oder l_2 Regularisierung, Dropout Srivastava et al. (2014) oder andere Regularisierungstechniken eingesetzt werden, steigt die Gefahr einer Überanpassung mit steigender Anzahl an Neuronen im Netzwerk.

	10000	1000	100	10	5	2
Test Fehler	62	19,9	14,9	14.9	14.9	15
Trainings Fehler	188	43,5	35,1	35,0	35,0	35,1

Anzahl an Iterationen

Die Anzahl der Iterationen, die benötigt werden, bis der Trainingsfehler nicht mehr sinkt, hängt stark von der gewählten Lernrate ab. Im weiteren Verlauf wird das einmalige Iterieren durch den gesamten Trainingsdatensatz als Epoche bezeichnet. In Abbildung 7 ist der Trainingsverlauf während mehrerer Epochen für verschiedene Lernraten abgebildet. Bei einer Lernrate von 0.1 sinkt der Trainingsfehler nach 100 Epochen nicht mehr.

Lernrate

Die Lernrate η reguliert die Auswirkung eines einzelnen Schrittes im Gradientenabstiegsverfahrens. Eine hohes η führt zu einer schnellen Minimierung des Trainingsfehlers. Jedoch steig die Gefahr, dass das Verfahren gute Minima überspringt, oder auch um ein gutes Minima Pendelt. Ein kleineres η findet mit einer hohen Wahrscheinlichkeit ein besseres Minima, jedoch werden dafür mehr Iterationen benötigt Günter Daniel Rey (2018).

In Abbildung 7 ist der Trainingsfehler während des Trainingsverlauf bei verschiedenen Lernraten dargestellt. Der Abbildung kann entnommen werden, dass für $\eta = 0.01$ das Training nach 300 Epochen immer noch nicht konvergiert ist. Für $\eta = 0.2$ ist zu sehen, dass das Verfahren sehr früh konvergiert und daher das Risiko besteht, dass wie oben beschrieben, gute Minima übersprungen wurden. Für komplexere Eingabedaten und Netzwerkarchitekturen würde dies ein reales Problem darstellen. In diesem einfachen Netzwerk und den einfachen Eingabedaten aus der Aufgabenstellung, kann auch mit $\eta = 0.2$ ein gutes Ergebnis erreicht werden. Der Mittelweg beider Extreme bildet $\eta = 0.1$ und wird als Einstellung vorgenommen.

Initialisierung der Gewichte

Vor Allem beim Trainieren von tiefen neuronalen Netzen spielt die Initialisierung der Gewichte eine entscheidende Rolle um das Auftreten des Problems der explodierenden beziehungsweise verschwindenden Gradienten entgegen zu wirken Geeron (2017). In Glorot and Bengio (2010) wird die Glorot-Initialisierung, ein Verfahren zur Initialisierung der Gewichte bei Verwendung der Sigmoid Aktivierungsfunktion, beschrieben. Hierbei

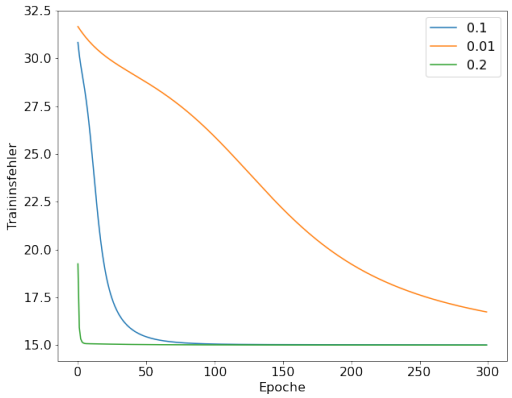

Abbildung 7: Trainingsfehler während der Trainingsepochen bei verschiedenen Lernraten

werden die Werte der initialen Gewichte aus einer Normalverteilung $\mathcal{N}(\mu, \sigma^2)$ mit $\mu = 0$ und $\sigma^2 = \frac{1}{fan_{avg}}$ entnommen. Hierbei gilt:

$$fan_{avg} = \frac{(fan_{in} + fan_{out})}{2}$$

Wobei fan_{in} der Anzahl an eingehenden Gewichte in einer Schicht entspricht und fan_{out} der Anzahl an Neuronen in der Schicht. Die Initialisierung der Gewichte in der vorliegenden Arbeit wurde nach Glorot implementiert.

Würden die Gewichte mit Null initialisiert werden, würde der Gradient für Alle Schichten bis auf die Ausgabeschicht Null sein. Ein Lernen würde nicht stattfinden. Der Gradient für die Aktualisierung der Gewichte zwischen versteckter Schicht und Ausgabeschicht würde für alle Partiellen Ableitungen gleich sein. Auch eine Initialisierung mit einer anderen Konstanten würde dazu führen, dass alle Gewichte äuf einer Ebene"gleich aktualisiert werden würden. Sie könnten einfach durch ein Neuron mit einer Verbindung ersetzt werden.

Mithilfe der Verwendung eines *Seeds* wird sicher gestellt, dass die zufällige Initialisierung der Gewichte bei jedem Durchlauf mit den gleichen Parametern die gleichen zufälligen Werte liefert. Das Netzwerk liefert also mit jedem Durchlauf bei gleichen Daten und Parametern die gleichen Ergebnisse. Somit wird die Reproduzierbarkeit im Netzwerk gefördert, was bei einer Fehlersuche und der Hyperparameter Optimierung hilfreich ist.

2.2 Neuronale Netze mit TensorFlow

Im Folgenden wird die Implementierung eines voll vernetzen Neuronalen Netzwerks mit Hilfe der *TensorFlow* Bibliothek beschrieben Abadi et al. (2015). Unterstützend wurde für den Aufbau des neuronalen Netzes die *Keras*-API verwendet, die seit *TensorFlow* Version 2 standardmäßig integriert ist.
Um die Daten für das Training des Neuronalen Netzes effizient vorzubereiten wurde die *Dataset*-API verwendet. Mit Hilfe der *Dataset*-API wurde das zufällige Mischen der Trainingsdaten, sowie die Bereitstellung in Form von *Mini-Batches* implementiert. Die Verwendung der *Dataset*-API hat noch zusätzlich den Vorteil, dass die Ausführung von Vorbereitungsschritten der Daten perfekt mit dem Training des Neuronalen Netzes koordiniert und parallelisiert werden können.

2.2.1 Normalisierung der Daten

2.2.2 Hyperparameter Suche

Um ein best Mögliches Modell für die vorliegenden Daten zu finden, wurde eine automatisierte Suche nach den besten Hyperparameter-Kombination implementiert. Die zur Auswahl stehenden Hyperparameter mit ihren möglichen Ausprägungen sind in Tabelle 1 aufgeführt.

Hyperparameter	**Ausprägungen**
Anzahl Hidden Layer	0,1,2,3,4
Anzahl Neuronen pro Schicht	1,3,5,10,20,50,100
Lernrate	0,1; 0,05; 0.01
Aktivierungsfunktion	ReLU, Sigmoid, ELU
Dropout Wahrscheinlichkeit	0; 0.25; 0.5

Tabelle 1: Learn rates during training the model.

Durch begrenzte Ressourcen wäre eine Evaluierung aller möglichen Hyperparameter-Kombinationen mittels einer *Grid Search* nicht möglich. In der vorliegenden Arbeit wurde deshalb eine zufällige Suche nach Bergstra and Bengio (2012) implementiert. Das Sieger-Modell aus 10 zufälligen Hyperparameter-Kombinationen htrainiat 4 versteckte Layer mit jeweils 3 Neuronen. Die Lernrate wurde auf 0.1 festgelegt und das Netzwerk wurde ohne die Verwendung von Dropout trainiert. Als effektivste Aktivierungsfunktion hat sich die ELU-Funktion herausgestellt Djork-Arné Clevert (2016). Das Sieger-Modell konnte eine Klassifizierungsgenauikgkeit von 85,11% auf die Testdaten erreichen. Die Daten zum trainieren und testen des neuronalen Netzwerks wurden analog zu Aufgabenstellung 1.3 verwendet.

3 Aufgabe 2: Ensemblemethoden

3.1 Unterkapitel 1

3.1.1 Unterkapitel 1.1

4 Aufgabe 4: Support Vector Machines

4.1 Unterkapitel 1

4.1.1 Unterkapitel 1.1

A Anhang

A.1 Ergänzende Abbildungen zu Booklet Teil 1

Folgende Abbildungen stellen die Entscheidungsbäume der Cost-Complexity Aufgabe dar. Der Unterschied in den Bäumen mit geringerem und höherem α wird für die beiden Baum-Variationen gut ersichtlich. Abbildungen 8 und 9 zeigen je einen vollständig ausgebildeten Baum, der durch Cost-Complexity „beschnitten“ wurden.

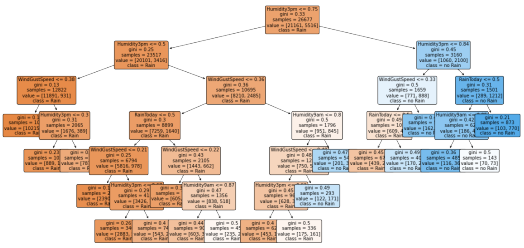

Abbildung 8: Entscheidungsbaum mit Parametern *max_ depth=None* und *ccp_ alpha=0.0005*

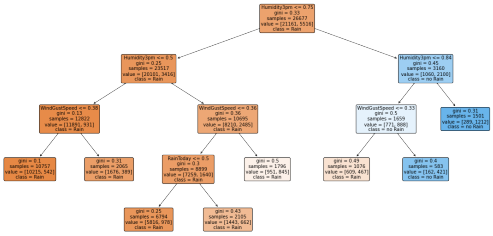

Abbildung 9: Entscheidungsbaum mit Parametern *max_ depth=None* und *ccp_ alpha=0.002*

Abbildungen 10 und 11 zeigen je einen beschnittenen Baum der mit der Einstellung *max_ depth = 8* erstellt wurde.

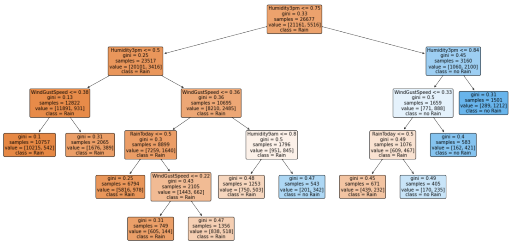

Abbildung 10: Entscheidungsbaum mit Parametern *max_ depth=8* und *ccp_ alpha=0.001*

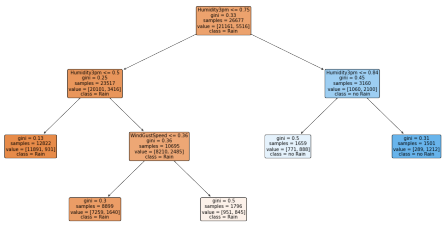

Abbildung 11: Entscheidungsbaum mit Parametern *max_ depth=8* und *ccp_ alpha=0.004*

A.2 Quellcode zu Booklet Teil 1

A.3 Quellcode zu Booklet Teil 2

A.4 Quellcode zu Booklet Teil 3

A.5 Quellcode zu Booklet Teil 4

Literatur

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, and Others (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Bergstra, J. and Y. Bengio (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research 13*(10), 281–305.

Djork-Arné Clevert, Thomas Unterthiner, S. H. (2016). Fast and accurate deep network learning by exponential linear units (elus).

Geeron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O'Reilly Media.

Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*.

Günter Daniel Rey, K. F. W. (2018). *Neuronale Netze*. hogrefe.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*(56), 1929–1958.