Projektarbeit

Certified Data Scientist

Ergebnisprädiktion mit grundlegenden und tiefer gehenden Machine Learning Algorithmen

verfasst im Rahmen der EN ISO / IEC 17024-Zertifizierungsprüfung von

Christian Koch

21.07.2021

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Projektarbeit eigenständig und ohne Mitwirkung Dritter angefertigt habe. Quellenangaben wurden entsprechend als solche gekennzeichnet.

Köln, 21.Juli 2021,

_____

Ort, Datum, Unterschrift

Inhalt

1Einleitung....4
1.1Aufgabe der Projektarbeit....5
1.2Aufbau der Projektarbeit....5
1.3Aufbau der Programmierarbeit....5
2Ausgangsdaten und deren Aufbereitung und Analyse....5
2.1Aufbau der Datenbank....5
2.2Erweiterung der Datenbasis....6
2.3Vorbereitung der Daten – Datenbank....6
2.4Vorbereiten der Daten – Programm....9
3Modell und Performance....10
3.1Modelle....10
3.1.1Neuronales Netz....13
3.2Performance....14
3.2.1Neuronales Netz....16
4Kritische Auseinandersetzung und Ausblick....18
5Literaturverzeichnis....19
6Abbildungsverzeichnis....20
7Anhang....20
7.1Anhang A: Beschreibung der Features....20
7.2Anhang B: Code des One-Hot-Encoding....23
7.3Anhang C: Datensatz Normalisierung und Encoding....23
7.4Anhang D: Code für Training des Neuronales Netzes....24

1 Einleitung

In praktisch allen Bereichen unseres Lebens werden heutzutage Daten gesammelt. Sämtliche Wirtschaftszweige häufen Daten an. Diese stehen zur Verfügung, um aus diesen nützliches Wissen und Informationen mittels geeigneter Methoden zu gewinnen. Die Leistungsfähigkeit von Computern, Netzwerken, die Verfügbarkeit von Frameworks für die Verknüpfung und Verwertung von Daten hat in den letzten Jahren enorm zugenommen. Dies eröffnet der Anwendung von Data-Science-Techniken die nötige Basis und hat die Nachfrage nach diesen deutlich gesteigert. [vgl. Provost, Fawcett 2017, S.23]

Data Science erlaubt es uns, Daten in nahezu beliebiger Menge zu analysieren und zu bearbeiten. Hierbei ist im besonderen das maschinelle Lernen („Machine Learning“) das primäre Werkzeug für das Erkennen von Mustern in Datensätzen und für daraus abgeleitete Prädiktionen. [vgl. Ng, Soo 2018, S.VIII]

Auch im Leistungssportbereich existiert eine hohe Bereitschaft, Daten auszuwerten und zu Zwecken der Verbesserung der Leistung zu verwenden, da hiermit häufig finanzielle Anreize verknüpft sind. [vgl. Papp et al 2019 ,S.250]

Sportvereine verwenden im Vergleich zu anderen Wirtschaftsbereichen nicht erst seit Big Data Datenanalysen zur zielführenderen Entscheidungsfindung, aber um die Jahrtausendwende gab es den ersten aufsehenerregenden Einsatz von Daten. Bekannt durch den gleichnamigen Film wurde der Begriff „Moneyball“ ein Synonym für den Erfolg eines Sportvereines, der vor allem durch die geschickte Verwendung und Verarbeitung von Daten erreicht wurde. [vgl. Memmert, Raabe 2019, S.55ff]

Im Fußball gibt es bereits seit einigen Jahren die ersten Vereine, die Daten ganz bewusst und gezielt in den Mittelpunkt ihrer Arbeit gestellt haben. So sind z.B. der dänische Verein FC Midtjylland und der vom selben Besitzer übernommene englische Verein FC Brentford Pioniere im Bereich des Profifußballs und damit Fußballvereine, die Daten zentral in ihrer Philosphie verankern und in verschiedensten Bereichen und von unterschiedlichen Typen einsetzen. So z.B. Ereignisdaten und in der letzten Zeit auch Positions- und Bewegungsdaten. Diese werden mit dem Ziel verwendet, insgesamt erfolgreicher zu sein, was durch Entscheidungen bei dem Zugang von neuen Spielern, dem Scouting, aber auch bei der Trainingssteuerung, der taktischen Ausrichtung und Entwicklung der Spieler zum Einsatz kommt. [vgl. Memmert, Raabe 2019, S.75ff]

1.1 Aufgabe der Projektarbeit

In dieser Arbeit wird auf Basis einer auf der Seite kaggle.com bereitgestellten Datensammlung (siehe Kapitel 2) der Versuch gewagt, Ergebnisse von Fußballspielen der Deutschen Fußballbundesliga der Saisons 2010/2011 bis 2015/2016 vorherzusagen.

Ob und wie gut dies gelungen ist, sowie die Auseinandersetzung mit ethischen Gesichtspunkten bei der Prädiktion von Sportergebnissen wird im Laufe und vor allem zum Abschluss der Arbeit diskutiert.

1.2 Aufbau der Projektarbeit

In den folgenden beiden Kapiteln wird die geleistete Programmierarbeit beschrieben.

Kapitel 2 beschreibt die Vorbereitung der Ausgangsdaten sowie deren Aufbereitung, sodass diese durch Modelle und im Training bzw. Test der Modelle angewandt werden können. Hier werden die Analysen der Daten dargestellt, z.B. zum Zwecke der Korrelationsprüfung.

Im Kapitel 3 werden die verwendeten Modelle vorgestellt und deren Parametrisierung. Zum Abschluss werden Aspekte der Performance der Modelle betrachtet.

Im letzten Kapitel 4 folgt eine kritische Auseinandersetzung mit den erzielten Ergebnissen sowie den digital ethischen Aspekten der Aufgabe. Außerdem wird vorausgeblickt, welche weiteren Möglichkeiten und Verbesserungen der Aufgabenstellung vorstellbar sind.

1.3 Aufbau der Programmierarbeit

Die Programmierarbeit ist zu finden in meinem Github-repository unter:
ChristianCKKoch/Projektarbeit_Digethic (github.com)

Die Struktur basiert auf der cookiecutter-Projektstruktur für Data Science-Projekte, was eine standardisierten und wiederverwendbaren Aufbau ermöglicht [Drivendata 2021, „Cookiecutter Data Science“]

In der README-Datei kann die genaue Projektstruktur eingesehen werden.

2 Ausgangsdaten und deren Aufbereitung und Analyse

Die Ausgangsdaten für diese Ausarbeitung stammen von der Seite kaggle.com [Mathien 2016, „European Soccer Database“] (Download der Datenbank am 2. Juli 2021) und liegen in einer SQLite-Datenbank vor.

2.1 Aufbau der Datenbank

Die Datenbank besteht aus den folgenden Tabellen:

Tabellenname	Inhalt
Country	Name und Id der 10 Länder, zu denen Spieldaten vorliegen
League	Name und Id und CountryId der 10 zu den jeweiligen Ländern gehörenden höchsten Spielklassen
Match	Detaillierte Informationen der Fußballbegegnungen der Saisons 2008/2009 bis einschließlich 2015/2016 aller 10 Ligen, u.a. • Saison, Datum, Spieltag • Heim-, Auswärtsmannschaft, Ergebnis • Startaufstellung inkl. Position • Ereignisse wie Freistöße, Eckbälle, etc. • Wettquoten der 10 führenden Wettanbieter
Player	Grundlegende Spielerinformation, wie Geburtstag, Größe, Gewicht
Player_Attributes	Aus dem EA Sportsspiel FIFA entnommene, monatliche Spielerattribute, wie Dribbling- oder Laufstärke. Nicht für alle Jahre verfügbar
Team	Mannschaftsnamen
Team_Attributes	Aus dem EA Sportsspiel FIFA entnommene, jährliche Attribute der Mannschaften zu Spielaufbau, Offensive und Defensive

Tabelle 1: Bestehende Datenbanktabellen

2.2 Erweiterung der Datenbasis

Diese Tabellen wurden noch um eine Tabelle erweitert: „Marktwert“. Am 8. Juli 2021 wurden die folgenden Daten von der Seite transfermarkt.de heruntergeladen:

Tabellenname	Inhalt
Markwert	Pro Mannschaft und Saison (2010-2015): • Kadergroesse • Durchschnittsalter • Legionaere • Durchschnittsmarktwert • Gesamtmarktwert

Tabelle 2: Hinzugefügte Datenbanktabelle "Martkwert"

2.3 Vorbereitung der Daten – Datenbank

Auf Basis der vorhanden Daten in der Datenbank wurden drei Datenbank-Views erstellt:

Viewname	Inhalt
vw_Match	Für die Deutsche Bundesliga und die Saisons 2010 bis 2015 werden hier die relevanten Daten für die Spieltagsergebnisse vereint und die Ergebnisse umgesetzt in Heimsieg, Unentschieden, Auswärtssieg
vw_Team_Attributes	In diesem View werden die Marktwerte und ausgewählte Attribute der Mannschaften pro Saison (2010 – 2015) vereint
vw_Match_Team_Data	Dies ist der zentrale View für die gestellte Aufgabe. Hier werden die ausgewählten Mannschaftwerte aus vw_Team_Attributes mit den Spieltagsdaten aus vw_Match kombiniert.

Tabelle 3: Erstellte Datenbank-Views

Die Eingangsdaten für die weiteren Schritte zum Trainieren und Auswerten der Modelle kommen aus dem View vw_Match_Team_Data und umfassen die in Anhang A: Beschreibung der Features beschriebenen Features.

Die absolute Verteilung der Zielvariable im Eingangsdatensatz stellt sich wie in der folgenden Abbildung dar:

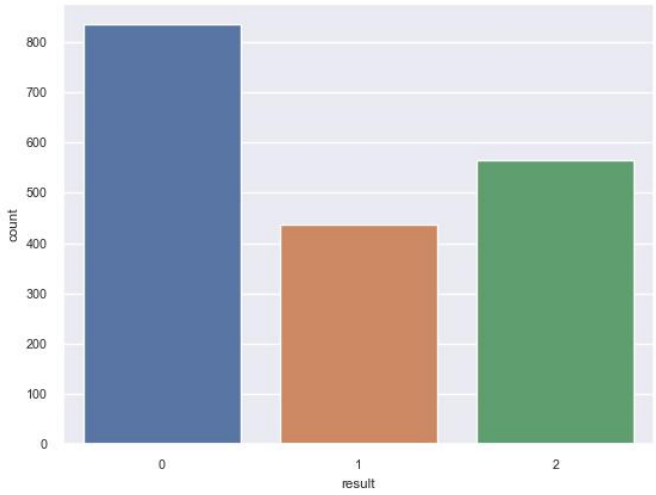

Abbildung 1: Absolute Verteilung der drei Zielkassen

- 0 = Heimsiege; absolute Anzahl: 835; relativ: 45,48%
- 1 = Unentschieden; absolute Anzahl: 437; relativ: 23,80%
- 2 = Auswärtssiege; absolute Anzahl: 564; relativ: 30,72%

Auf Basis einer Korrelationsmatrix konnte der Einfluss der Features auf das Resultat sowie untereinander ermittelt werden:

Wirklich starke Korrelationen untereinander sind nicht zu registrieren, außer, dass sich die Team-Werte im Laufe der Saisons verändern.

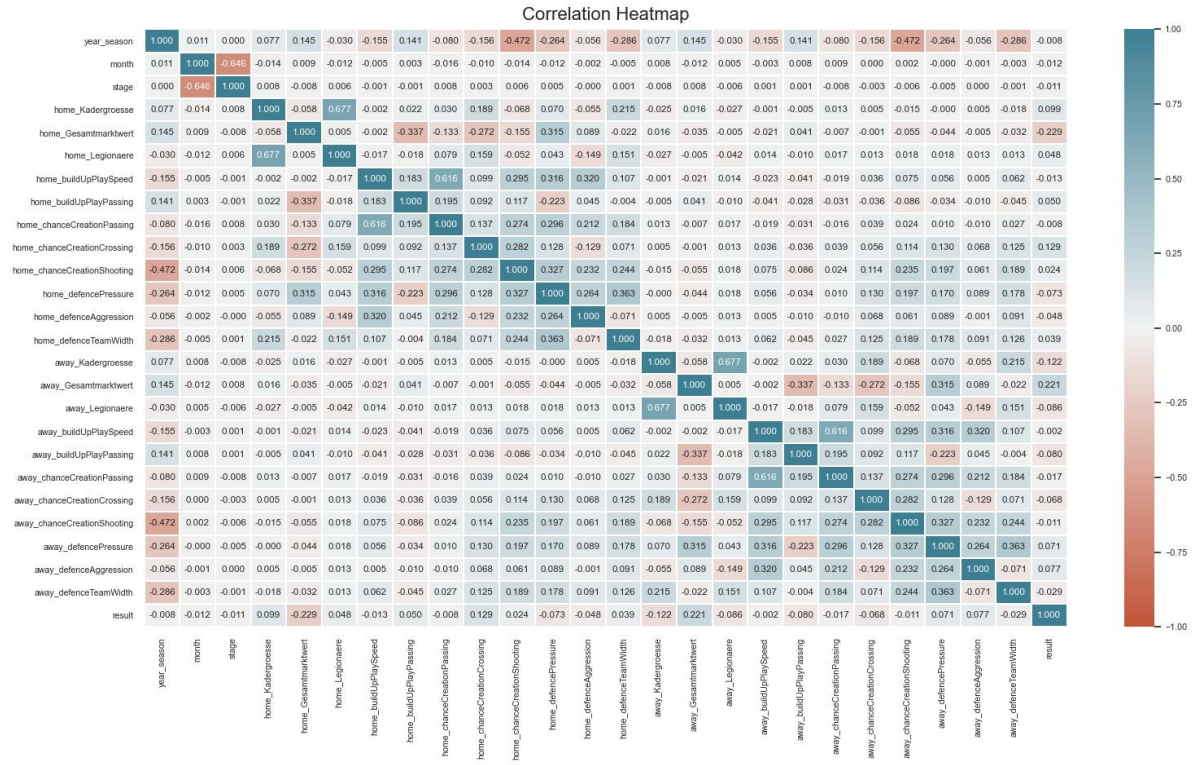

Abbildung 2: Heatmap aller Korrelationen der Eingangsdaten untereinander

Eine Abhängigkeit der Zielvariable „result“ bzw. signifikanter Einfluss bestimmter Variablen ist jedoch gut zu erkennen:

Feature	Korrelationskoeffizient
result	1.000000
home_Gesamtmarktwert	0.228897
away_Gesamtmarktwert	0.220900
home_chanceCreationCrossing	0.129018
away_Kadergroesse	0.121751
home_Kadergroesse	0.098924
away_Legionaere	0.086038
away_buildUpPlayPassing	0.080120
home_defencePressure	0.073091
away_defencePressure	0.070710
away_chanceCreationCrossing	0.068214
home_buildUpPlayPassing	0.050309
home_defenceAggression	0.048356
home_Legionaere	0.047563
home_defenceTeamWidth	0.038687
away_defenceTeamWidth	0.029374
home_chanceCreationShooting	0.023738

Feature	Korrelationskoeffizient
away_chanceCreationPassing	0.017030
home_buildUpPlaySpeed	0.012662
month	0.012486
stage	0.011454
away_chanceCreationShooting	0.010868
home_chanceCreationPassing	0.007964
year_season	0.007599
away_buildUpPlaySpeed	0.002458

Tabelle 4: Lister der Korrelationskoeffizienten der Features in Bezug auf die Zielvariable

Spieltagsnummer und Monat werden herausgenommen, da wenig bis keine Auswirkung auf die Zielvariable besteht. Diese beiden Features würden eine Prädiktionsanwendung stark einschränken, da diese immer mit angegeben werden müssten. So kann eine Ergebnisvorhersage auf Basis der folgenden Angaben

- Heimmannschaft
- Auswärtsmannschaft
- Die betreffende Saison

durchgeführt werden. Alle Team-Werte und -Attribute können mittels Datenbankanfragen hinzugefügt werden.

Der Datenbank-View vw_Match_Team_Data ist entsprechend angepasst und die beiden Felder „stage“ und „month“ entfernt worden.

2.4 Vorbereiten der Daten – Programm

Nachdem die Eingangsdaten soweit vorbereitet sind, werden diese nun für die Verwendung in den Machine-Learn-Algorithmen umgewandelt.

Hierzu wird im ersten Schritt ein One-Hot-Encoding eingesetzt, um die Namen der Mannschaften umzusetzen, siehe Anhang Anhang B: Code des One-Hot-Encoding für den detaillierten Code.

In weiteren Schritten werden die Spalten mit den ursprünglichen Mannschaftsnamen entfernt und der Datensatz getrennt in Features und die Zielklassifikation.

Mittels der Preprocesser-Funktionen (preprocessor.py) werden die Daten nun gesplittet in Trainings- und Testdatensätze. Hierbei werden die Daten auch gleich mittels MinMaxScaler normalisiert.

Siehe Anhang C: Datensatz Normalisierung und Encoding für drei Screenshots mit den Daten vor und nach Normalisierung sowie das Ergebnis des One-Hot-Encodings.

Die Daten werden in einer Pickle-Datei abgespeichert im Unterordner /data/processed und in den nächsten Schritten weiterverwendet.

3 Modell und Performance

Die grundlegende Idee und Funktionsweise der erarbeiteten Lösung ist, dass in einer Modellbibliothek die Modelle definiert werden und aus einem Programmaufruf dann beliebig in- oder exkludiert werden. Alle inkludierten Modelle werden auf Basis der Trainingsdaten trainiert und anhand der Testdaten bewertet (einfache Akkuranzbestimmung).

Bei den Modellen wird in klassischen und Deep-Learning-Modellen unterschieden. Die klassischen Modelle sind aufgrund der Aufgabenstellung allesamt Klassifizierungsmodelle.

Auf Basis der Akkuranzbestimmung wird dann das beste („best performende“) Modell ermittelt und in einer Pickle-Datei abgelegt, sodass dies in einer Prädiktionsanwendung Verwendung finden kann.

Für dieses beste Modell werden noch weitere Performance-Analysen durchgeführt, im Falle des Neuronalen Netzes auch noch die Darstellung des Overfitting und der angewandten Fehlerfunktion.

Die folgenden Machine-Learning-Modelle sind implementiert:

- KNeighbors Classifier (sklearn.neighbors)
- Decision Tree Classifier (sklearn.tree)
- Random Forest Classifier (sklearn.ensemble)
- Support Vector Machine (SVC) (sklearn.svm)
- MLP Classifier (sklearn.neural_network)
- Voting Classifier (sklearn.ensemble)
- Neuronales Netzwerk (Pytorch auf Basis von torch.nn.Module)

3.1 Modelle

In der folgenden Tabelle werden die implementierten Modelle kurz erläutert mit dem Fokus auf deren Parameterisierung. Das Neuronale Netz, das mittels Pytorch erstellt wurden, wird gesondert beschrieben.

Modell	Beschreibung
KNeighbors Classifier	Der KNeighbors Classifier gehört zu den Instanz-basierten Algorithmen im Machine Learning.

	Die im Rahmen dieser Arbeit vorgenommene Optimierung beschränkt sich auf den Parameter für die Anzahl der Nachbarn (n_neighbors). Hierzu wird eine selbstgebaute Search-Optimierung verwendet, die von 1 bis einschließlich 250 aufsteigend alle Werte für n_neighbors prüft und mittels einer Fehlerfunktion (Gemittelte Abweichung des erwarteten vom tatsächlichen Ergebnis) den optimalen Wert bestimmt.
	
	`#-----------------------`
	`#Knn-Classifier`
	`#-----------------------`
	`if model == 'knn':`
	`  #Optimalen Knn-Classifier bestimmen`
	`  error = []`
	`  for i in range(1, 250):`
	`  knn = KNeighborsClassifier(n_neighbors=i)`
	`  knn.fit(self.X_train, self.y_train)`
	`  pred_i = knn.predict(self.X_test)`
	`  error.append([i, np.mean(pred_i != self.y_test)])`
	
	In den durchgeführten Durchläufen ergab sich meist ein n=122 als optimale Anzahl Nachbarn, mit einer Fehlerquote von 0.453804347826087.

Decision Tree Classifier	Der Decision Tree ist Modell-basiert und gehört zu den Baummethoden.
	
	Zur Optimierung der Hyperparameter wird GridSearch eingesetzt. Die in dieser Arbeit optimierten Parameter sind:
	
	• criterion – Das Kriterium für die Bestimmung des Information Gain
	• max_depth – Die maximale Baumtiefe
	• min_samples_split – Die minimale Anzahl der Datensätze zum erstellen neuer interner Knoten des Baumes
	
	Mehr Information vgl. scikit-learn developers (2007 – 2020), „sklearn.tree.DecisionTreeClassifier“
	
	`#Optimalen Decision Tree bestimmen`
	`#Zu testende Decision Tree Parameter`
	`dt = DecisionTreeClassifier()`
	`tree_para = {'criterion':['gini','entropy'],'max_depth':[i for i in range(1,20)]`
	`  , 'min_samples_split':[i for i in range (2,10)]}`
	
	`#GridSearchCV`
	`grd_clf = GridSearchCV(dt, tree_para, cv=5)`
	`grd_clf.fit(self.X_train, self.y_train)`
	
	`#Besten gefundenen Decision Tree übergeben`
	`dt_clf = grd_clf.best_estimator_`
	
	Die häufig beobachtete optimale Kombination der Hyperparameter war:
	
	best parameters: {'criterion': 'gini', 'max_depth': 2, 'min_samples_split': 2}

Random Forest Classifier	Der Random Forest Classifier gehört zu den Ensemble-Learning-Modellen. Im Prinzip werden n Anzahl Decision Trees erstellt und das Ergebnis demokratisch über diese verschiedenen Bäume abgestimmt und bestimmt. Eine Optimierung wird hier nicht vorgenommen, der Random Forest soll 100 Bäume verwenden (n_estimators = 100)

Support Vector Machine	Für die Support Vector Machine (SVM) werden zwei Hyperparameter gesetzt: • kernel – Die zu verwendende Methode für den Kernel-Trick; in diesem Fall wird 'poly' (Polynomialer Kernel) verwendet • probability – diese wird auf 'True' gesetzt, sodass die SVM neben den Ergebnissen auch die Rückgabe der probabilistischen Wahrscheinlichkeiten für jede Klasse zurückgibt
MLP Classifier	Der MLP (Multi-layer Perceptron) Classifier ist ein Klassifizierer auf Basis eines neuronalen Netzwerks. Auf Basis von Probieren und Testen sind die folgenden Parameter-Werte gesetzt worden: • hidden_layer_sizes – Im Rahmen von 100 layern und 100 Neuronen kann der Klassifizierer seine Struktur selbt optimal bestimmen • max_iter – Die maximale Anzahl von Epochen wurden auf 500 gesetzt • solver – Als Optimierer wird 'sgd' (stochastic gradient descent) verwendet • learning_rate – Die Lernrate des Optimierers soll adaptiv sein, sich also dynamisch anpassen, um ein Überschießen über das Minimum hinaus zu verhindern • learning_rate_init – Die Lernrate wird mit dem Wert 0.01 initialisiert • n_iter_no_change – Die maximale Anzahl der Epochen, in denen eine Verbesserung der Fehlerfunktion einen Toleranzwert (1e-4) nicht mehr überschreitet. Danach wird das Training automatisch gestoppt • early_stopping – Wird auf True gesetzt, damit n_iter_no_change greifen kann `mlp = MLPClassifier(hidden_layer_sizes=[100,100], max_iter=5000, solver='sgd' , learning_rate='adaptive', learning_rate_init=0.01, n_iter_no_change=200, early_stopping=True)`

Voting Classifier	Der Voting Classifier gehört ebenfalls wie der Random Forest zu den Ensemble-Learning-Methoden.
	Es werden alle inkludierten Methoden, also maximal alle hierüber beschriebenen Methoden verwendet für den Voting Classifier. Dieser bestimmt auf Basis der probabilistischen Klassenwahrscheinlichkeit über alle Methoden hinweg das beste Ergebnis einer Klassifizierung
	`def ensemble_model(self):`
	`  #Alle inkludierten Modelle werden in eine Liste geladen, die dann als Parameter`
	`  #dem Voting Classifier übergeben wird.`
	`  models = list()`
	`  for model in self.ergebnis:`
	`  models.append([model[0], model[2]])`
	`  voting_clf = VotingClassifier(estimators=models, voting='soft')`

3.1.1 Neuronales Netz

Das Neuronale Netz, das mittels Pytorch erstellt wird, wird an dieser Stelle etwas ausführlicher beschrieben.

Es ist wie alle anderen Modelle in der Modellbibliothek model-library.py zu finden.

Für die Definition des neuronalen Netzes wird das Pytorch Modul „torch.nn.Module“ verwendet und von diesem geerbt:

```
class NN_Model(torch.nn.Module):
    def __init__(self):
        super(NN_Model, self).__init__()
        self.fc1 = nn.Linear(75,120)
        self.fc2 = nn.Linear(120,180)
        self.fc3 = nn.Linear(180,100)
        self.fc4 = nn.Linear(100,40)
        self.output = nn.Linear(40,3)

    def forward(self,x):
        x = torch.relu(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.sigmoid(self.fc4(x))
        #Keine Softmax-Funktion benötigt bei output, da CrossEntropyLoss
        #als Fehlerfunktion dies automatisch tut
        #Bemerkung: softmax muss aber beim Validieren/Testen angewandt werden!
        x = self.output(x)

        return x
```

Abbildung 3: Code Definition des Neuronalen Netzes

Es besteht also aus einem Eingangslayer mit 75 Neuronen (=Anzahl Features), aus vier Hidden Layers (120, 180, 100 und 40 Neuronen) und einer Ausgabe-Schicht mit den drei Zielklassen. Es wird keine Aktivierungsfunktion verwendet für die Ausgabe-Schicht. Dies liegt daran, dass CrossEntropyLoss verwendet wird und diese Fehlerfunktion selbst eine Softmax-Funktion (in diesem Falle LogSoftmax) anwendet auf den Ausgabe-Tensor.

Für den Aufruf und das Trainieren des Neuronalen Netzes ist die Funktion „neuronal_network(self, epochs, patience_early_stopping, threshold_for_early_stopping)“ erstellt worden.Siehe Anhang D: Code für Training des Neuronales Netzes für das komplette Listing dieser Funktion.

Hier die wichtigsten Elemente der Funktionsweise:

- Die gewählte Fehlerfunkion ist CrossEntropyLoss. Diese ist geeignet für Klassifizierungen mit mehr als zwei Ergebnisklassen (vgl. Torch Contributors (2019), „CROSSENTROPYLOSS“)
- Der gewählte Optimizer ist Adam. Dieser eignet sich für Multiklassen-Klassifikationen und erzielt perfomant gute Ergebnisse (vgl. vgl. Torch Contributors (2019), „ADAM“)
- Die maximale Anzahl der Epochen wird als Parameter übergeben

- Es ist ein Early Stopping implementiert. Dieses prüft anhand der Veränderung der Lossfunktion des Trainingsdatensatzes, ob ein Early Stop, also das Unterbrechen der Epochen, sinnvoll ist. Für das Early Stopping werden die folgenden Parameter übergeben
    - patience – Die Anzahl der Epochen, die abwartet werden soll, wenn das Ergebnis der Fehlerfunktion der Testdaten NICHT mehr sinkt (hier könnte optional auch die Sinkrate angegeben werden. Diese wird auf dem Default-Wert belassen, da dies die strengste, also kleinste Veränderung bedeutet)
    - threshold_acc – Der Threshold für die Akkuranz. Hierdurch wird überhaupt erst ab einer bestimmten Akkuranz das Early Stopping aktiviert. Durch Erfahrung mit dem Neuronalen Netz kann hier ein deutlich sinnvolleres Verhalten des Early Stopping und dadurch höherer Performance erreicht werden.
- Während des Durchlaufens der Epochen wird ständig die Akkuranz des Trainingsdatensatzes, des Testdatensatzes sowie des Fehlers auf Basis der Fehlerfunktion mitgeschrieben und in Arrays gespeichert, die als Rückgabewerte an den Aufruf übergeben werden.

Dass hier kein eigener Validierungsdatensatz verwendet wird, kann kritisch angemerkt werden, wurde jedoch bewusst so gewählt, da der gesamte Eingangsdatensatz nicht besonders groß ist und somit ein verhältnismäßig großer Datensatz für das Training zur Verfügung steht.

3.2 Performance

Wie bereits zu Eingang dieses Kapitels beschrieben, wird das Modell mit der besten Performance (auf Basis von Akkuranz) schlussendlich gespeichert in einer Pickle-Datei und kann somit in einer Prädiktionsanwendung benutzt werden.

Nach einem Durchlauf wird in einer Tabelle das Ergebnis-Array gespeichert:

Name des Modells	Akkuranz	Modellobjekt
KNeighborsClassifier	54,62%	KNeighborsClassifier(n_neighbors=122)
VotingClassifier	52,72%	VotingClassifier(estimators=[['RandomForestClassifier', RandomForestClassifier()], ['KNeighborsClassifier', KNeighborsClassifier(n_neighbors=122)], ['SVC', SVC(kernel='poly', probability=True)], ['DecisionTreeClassifier', DecisionTreeClassifier(max_depth=2)], ['MLPClassifier', MLPClassifier(early_stopping=True, hidden_layer_sizes=[100, 100], learning_rate='adaptive', learning_rate_init=0.01, max_iter=5000, n_iter_no_change=200, solver='sgd')]], voting='soft')
NN_Model	52,17%	NN_Model((fc1): Linear(in_features=75, out_features=120, bias=True) (fc2): Linear(in_features=120, out_features=180, bias=True) (fc3): Linear(in_features=180, out_features=100, bias=True) (fc4): Linear(in_features=100, out_features=40, bias=True) (output): Linear(in_features=40, out_features=3, bias=True))
SVC	51,90%	SVC(kernel='poly', probability=True)
MLPClassifier	51,63%	MLPClassifier(early_stopping=True, hidden_layer_sizes=[100, 100], learning_rate='adaptive', learning_rate_init=0.01, max_iter=5000, n_iter_no_change=200, solver='sgd')
RandomForestClassifier	51,36%	RandomForestClassifier()
DecisionTreeClassifier	49,18%	DecisionTreeClassifier(max_depth=2)

Abbildung 4: Ergebnis-Tabelle mit allen implementierten Modellen

In diesem Fall war der KNeighbors-Classifier das Modell mit der höchsten Akkuranz im Test.

Das Programm gibt dementsprechend aus:

Bestes Modell ist: KNeighborsClassifier mit einer Akkuranz von 54,61956521739131%

Dieses Modell kann also in gut 54% der Fälle das korrekte Endergebnis eines Bundesliga-Spiels in den Saisons 2010-2015 vorhersagen. Das ist besser als die Base Rate von 45,48% (siehe 2.3). Die Ergebnisse werden in Kapitel 4 noch detaillierter betrachtet.

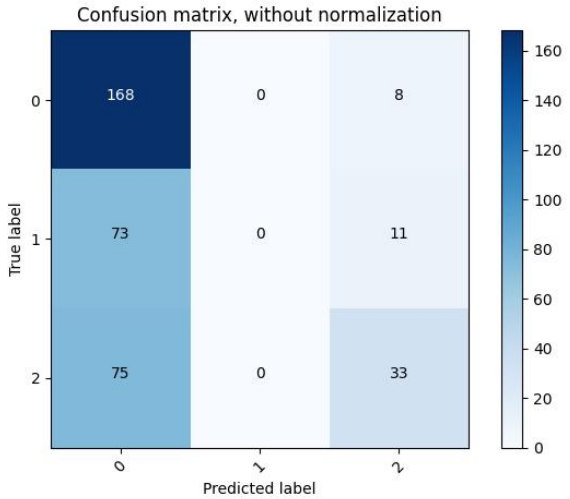

Abbildung 5: KNeighbours - Confusion Matrix absolut

Die Confusion Matrix zeigt ein interessantes Bild: Es werden keine Unentschieden vorhergesagt. Und auch die Anzahl der Auswärtssiege ist deutlich niedriger als die Erwartung. Erwartet wird in etwa die Verteilung im Eingangsdatensatz, siehe Kapitel 2.3.

Dies erscheint jedoch sinnvoll, da die Wahrscheinlichkeit eines Heimsieges am höchsten ist. Dennoch wird dieses Modell wohl beinahe niemals ein Unentschieden voraussagen können.

3.2.1 Neuronales Netz

An dieser Stelle soll die Performance des Neuronalen Netzes nochmals etwas ausführlicher betrachtet werden. Es erzielt bzgl. der Akkuranz nicht die guten Werte eines KNN, ist jedoch unter den besten Modellen zu finden.

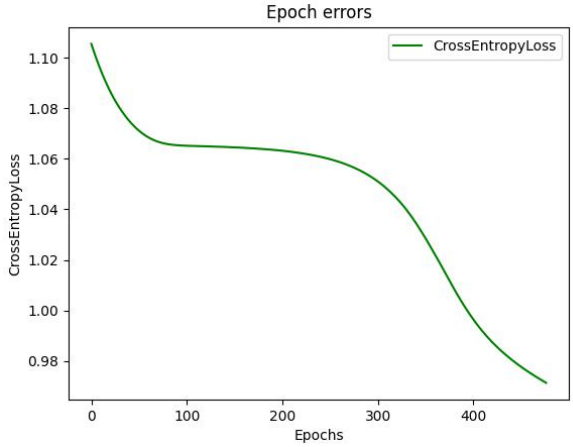

Abbildung 6: Werte der Fehlerfunktion im Laufe der Epochen

Man kann sehen, dass das Trainieren funktioniert. Die Werte der Fehlerfunktion werden im Laufe der Epochen minimiert.

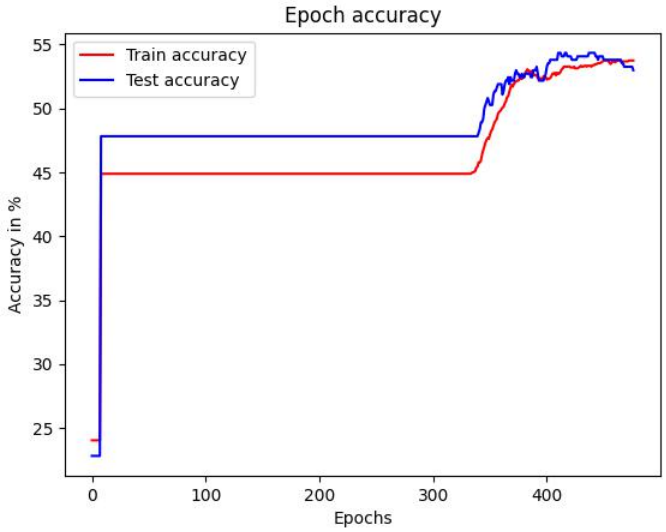

Abbildung 7: Veränderung der Akkuranz des Trainings- und Testdatensatzes im Laufe der Epochen

Die Betrachtung der Akkuranz im Laufe der Epochen lässt das Lernverhalten des Neuronalen Netzes sehen. Zunächst passiert viele Epochen lang nicht viel, dann setzt durch die stetige Minimierung des Fehlers das Lernen ein. Die Akkuranz in beiden Datensätzen, dem Trainings- als auch dem Testdatensatz nimmt stetig zu.

In diesem Durchlauf sind insgesamt 476 Epochen durchrechnet worden. Dann hat die Early-Stopping-Logik das weitere Durchlaufen unterbrochen. Die Akkuranz bzw. die Fehlerwerte von Trainings- und Testdaten drohten auseinanderzudriften. Hierdurch wird ein Overfitting des Modells erkannt und dementsprechend verhindert.

```
Epoche: 466/1000 mit Train-Akkuranz: 53.678474114441414 und Test-Akkuranz: 53.53260869565217
Epoche: 467/1000 mit Train-Akkuranz: 53.678474114441414 und Test-Akkuranz: 53.53260869565217
Epoche: 468/1000 mit Train-Akkuranz: 53.678474114441414 und Test-Akkuranz: 53.26086956521739
Epoche: 469/1000 mit Train-Akkuranz: 53.678474114441414 und Test-Akkuranz: 53.26086956521739
Epoche: 470/1000 mit Train-Akkuranz: 53.678474114441414 und Test-Akkuranz: 53.26086956521739
Epoche: 471/1000 mit Train-Akkuranz: 53.678474114441414 und Test-Akkuranz: 53.26086956521739
Epoche: 472/1000 mit Train-Akkuranz: 53.746594005449595 und Test-Akkuranz: 53.26086956521739
Epoche: 473/1000 mit Train-Akkuranz: 53.746594005449595 und Test-Akkuranz: 53.26086956521739
Epoche: 474/1000 mit Train-Akkuranz: 53.746594005449595 und Test-Akkuranz: 53.26086956521739
Epoche: 475/1000 mit Train-Akkuranz: 53.746594005449595 und Test-Akkuranz: 53.26086956521739
EarlyStopping counter: 1 out of 2
Epoche: 476/1000 mit Train-Akkuranz: 53.746594005449595 und Test-Akkuranz: 52.98913043478261
EarlyStopping counter: 2 out of 2
Early stopping
Bestes Model ist: NN_Model mit einer Akkuranz von 52.9891304347826%
```

Abbildung 8: Ausschnitt der Programmausgabe beim Neuronalen Netz

Auch für das Neuronale Netz wird eine Confusion-Matrix erstellt:

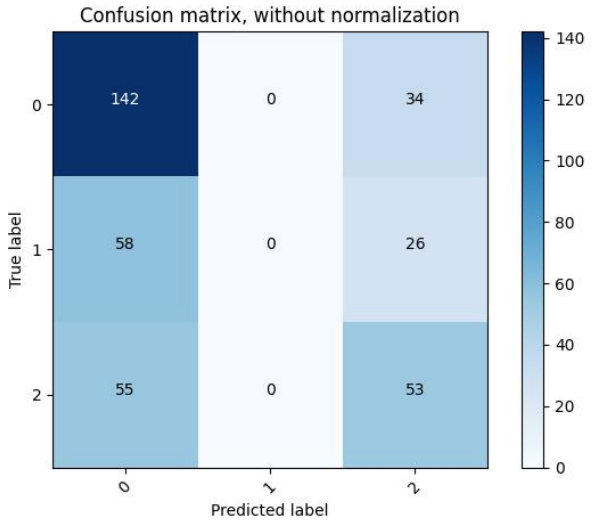

Abbildung 9: Neuronales Netz - Confusion Matrix absolut

Hier zeigt sich ein ähnliches Bild wie bei dem KNeighbor-Modell. Im Gegensatz zu diesem gibt das Neuronale Netz aber deutlich mehr Auswärtssiege aus und kommt damit der erwarteten Verteilung (siehe Kapitel 2.3) näher.

4 Kritische Auseinandersetzung und Ausblick

Bei der kritischen Auseinandersetzung mit der erbrachten Leistung möchte ich zunächst auf die erzielten Ergebnisse eingehen und dann im Anschluss ethische Aspekte betrachten sowie einen Ausblick geben.

Die erzielten Ergebnisse sehen auf den ersten Blick nicht besonders erfolgversprechend aus. Eine Akkuranz von ca. 54% und eine Confusion Matrix, die darauf hindeutet, dass so gut wie in keinem Fall ein Unentschieden vorhergesagt werden kann. Doch bei näherer Betrachtung entspricht dies durchaus einem zu erwartenden guten Ergebnis:

Der Autor des Ausgangsdatensatzes beschreibt in der Herausforderung, dass das zu „schlagende“ Ergebnis die Quote der Buchmacher ist, die bei 53% liege. Er selber kommt mit einem Support-Vector-Machine-Modell auf ebenso 53%. [vgl. Mathien 2016, „European Soccer Database“].

Auch die Base Rate, die in diesem Fall als Orientierung herangezogen werden kann (und nicht wie in einem klassischen binären Klassifizierungsfall als Benchmark gilt) liegt mit 45,48% unter den erzielten Akkuranz-Ergebnissen und zeigt, dass die erstellten Modelle besser sind als stupides, stetiges „Setzen“ auf Heimsiege.

Der Einsatz von Ergebnisvorhersagen sollte meiner Meinung nach im besten Falle zur Leistungsbeurteilung als eine von vielen Komponenten einer zusätzlich Daten getriebenen Spielanalyse und vor allem Spielvorbereitung im Fußball dienen.

Die erstellten Methoden könnten auch eingesetzt werden, um Missbrauch bzw. Betrug im Wettgeschäft aufzudecken. Welche Quoten erscheinen unberechtigt hoch? Welche unplausiblen und dadurch auffälligen Ergebnisse haben bestimmte Mannschaften entgegen der Erwartung erzielt?

Die Möglichkeiten von Data-Science-Methoden können natürlich auch in die falschen Hände geraten. So könnte z.B. die „Wettmafia“ bessere Wege finden, noch gezielter und damit versteckter Spiele zu manipulieren. Auf diese Weise könnten genau solche wie im vorigen Abschnitt erwähnten Auffälligkeiten umgangen und trotzdem große Gewinne erzielt werden.

In der vorliegenden Arbeit ist zum Zeitpunkt der Abgabe nur eine sehr simple Form der Prädiktion erstellt worden. Diese könnte in einem ersten Schritt erweitert werden, wie auf Seite 9 diskutiert. Dann wäre es weiterhin interessant, die Methoden weiterzuentwickeln, erweiterte Optimierung der Hyperparameter einzusetzen und auch weitere Datenquellen miteinzubeziehen. Auch die zeitliche Komponente, die hier aufgrund der geringen Jahre außer Acht gelassen wurde, könnte hinzugezogen werden.

5 Literaturverzeichnis

Provost, Foster, Fawcett, Tom (2017), Data Sience für Unternehmen – Data Mining und datenanalystisches Denken praktisch anwenden, 1. Auflage, mitp Verlags GmbH & Co KG, Frechen

Ng, Annalyn, Soo, Kenneth (2018, Data Science – was ist das eigentlich?, 1. Auflage, Springer Verlag, Berlin

Papp, Stefan et al (2019), Handbuch Data Science – Mit Datananalyse und Machine Learning Wert aus Daten generieren, 1. Auflage, Carl Hanser Verlag, München

Memmert, Daniel, Raabe, Dominik (2018), Revolution im Profifußball – Mit Big Data zur Spielanalyse 4.0, 2. Auflage, Springer Verlag, Berlin

Hugo Mathien (2016), „European Soccer Database“, https://www.kaggle.com/hugomathien/soccer, letzter Zugriff am 20.07.2021

Drivendata (2021), „Cookiecutter Data Science“, https://drivendata.github.io/cookiecutter-data-science/, letzter Zugriff am 03.07.2021

scikit-learn developers (2007 – 2020), „sklearn.tree.DecisionTreeClassifier“, https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html, letzter Zugriff am 14.07.2021

Torch Contributors (2019), „CROSSENTROPYLOSS“, https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html, letzter Zugriff am 19.07.2021

Torch Contributors (2019), „ADAM“, https://pytorch.org/docs/stable/generated/torch.optim.Adam.html, letzter Zugriff am 19.07.2021

6 Abbildungsverzeichnis

Abbildung 1: Absolute Verteilung der drei Zielkassen, Seite 7
Abbildung 2: Heatmap aller Korrelationen der Eingangsdaten untereinander, Seite 8
Abbildung 3: Code Definition des Neuronalen Netzes, Seite 13
Abbildung 4: Ergebnis-Tabelle mit allen implementierten Modellen, Seite 15
Abbildung 5: KNeighbours - Confusion Matrix absolut, Seite 15
Fehler: Referenz nicht gefunden, Seite Fehler: Referenz nicht gefunden
Abbildung 7: Veränderung der Akkuranz des Trainings- und Testdatensatzes im Laufe der Epochen, Seite 16
Abbildung 8: Ausschnitt der Programmausgabe beim Neuronalen Netz, Seite 17
Abbildung 9: Neuronales Netz - Confusion Matrix absolut, Seite 17
Abbildung 10: Eingangsdaten (Auszug) VOR Normalisierung, Seite 21
Abbildung 11: Eingangsdaten (Auszug) NACH Normalisierung, Seite 22
Abbildung 12: Ergebnis des One-Hot-Encoding (Auszug), Seite 22

7 Anhang

7.1 Anhang A: Beschreibung der Features

Beschreibung der Features

Feature	Kurze Beschreibung	Beispielwert
season	Saison, in der die Begegnung stattfand	2011
month	Monat, in der die Begegnung stattfand	4
stage	Spieltagsnummer	32
home_team_name	Name der Heimmannschaft	1. FC Köln
away_team_name	Name der Auswärtsmannschaft	VfB Stuttgart
home_Kadergroesse / away_Kadergroesse	Kadergröße der Heim-/ Auswärtsmannschaft	31 / 32
home_Gesamtmarktwert / away_Gesamtmarktwert	Gesamtmarktwert der Heim-/ Auswärtsmannschaft in Mio. €	69.29 / 101.88
home_Legionaere / away_Legionaere	Anzahl der Legionäre in der Heim-/ Auswärtsmannschaft	17 / 18
home_buildUpPlaySpeed / away_buildUpPlaySpeed	FIFA-Spiel Bewertung der Geschwindigkeit des Aufbauspiels der Heim-/ Auswärtsmannschaft	58 / 64
home_buildUpPlayPassing / away_buildUpPlayPassing	FIFA-Spiel Bewertung des Passspiels im Aufbauspiel der Heim-/ Auswärtsmannschaft	71 / 48
home_chanceCreationPassing / away_chanceCreationPassing	FIFA-Spiel Bewertung des Passspiels in der Offensive der Heim-/ Auswärtsmannschaft	42 / 60
home_chanceCreationCrossing / away_chanceCreationCrossing	FIFA-Spiel Bewertung des Querspiels in der Offensive der Heim-/ Auswärtsmannschaft	39 / 62
home_chanceCreationShooting / away_chanceCreationShooting	FIFA-Spiel Bewertung der Schussqualität in der Offensive der Heim-/ Auswärtsmannschaft	50 / 56
home_defencePressure / away_defencePressure	FIFA-Spiel Bewertung der Druckerzeugung in der Defensive der Heim-/ Auswärtsmannschaft	40 / 45
home_defenceAggression / away_defenceAggression	FIFA-Spiel Bewertung der Aggressivität in der Defensive der Heim-/ Auswärtsmannschaft	40 / 56
home_defenceTeamWidth / away_defenceTeamWidth	FIFA-Spiel Bewertung der Weite in der Defensive der Heim-/ Auswärtsmannschaft	56 / 39
result	Das Resultat der Begegnung.	1

	0 – Heimsieg 1 – Unentschieden 2 - Auswärtssieg	

7.2 Anhang B: Code des One-Hot-Encoding

Ausschnit aus prepare.py für das One-Hot-Encoding:

```
#Teamname-Spalten encoden (one-hot-encoding)

for spalte in ('home_team_name','away_team_name'):
  team_name = daten[spalte]
  #Anpassung der Form des Vektors mit den Werten der Teamnamen
  team_name_res = team_name.values.reshape(-1,1)
  #Initialisierung des One Hot Encoders
  team_name_onehot_enc = OneHotEncoder()
  #Fit des One Hot Encoders
  team_name_onehot = team_name_onehot_enc.fit_transform(team_name_res)

  #Die in alphabetischer Reihenfolge angeordneten Teamnamen werden in einen Array geladen
  sp_name_array = team_name_onehot_enc.categories_[0]
  #Dieser Array wird durchlaufen und die Teamnamen im Falle der Heimteams um die Endung "_H" bzw.
  #im Falle der Auswärtsteams durch die Endung "_A" erweitert
  for i in range(0, sp_name_array.size):
    sp_name_array[i] = sp_name_array[i] + '_H' if spalte == 'home_team_name' else sp_name_array[i] + '_A'

  #Zusammenführen (Join) der One-hot-encodeten Team-Namen mit dem ursprünglichen Datensatz
  daten_join = daten_join.join(pd.DataFrame(team_name_onehot.toarray(), columns=sp_name_array))

result_spalte = daten_join.pop("result")
daten_join['result'] = result_spalte
daten_prepared = daten_join.drop(["home_team_name", "away_team_name"], axis=1)
```

7.3 Anhang C: Datensatz Normalisierung und Encoding

	year_season	home_Kadergroesse	home_Gesamtmarktwert	home_Legionaere	home_buildUpPlaySpeed	home_buildUpPlayPassing	l
0	2010	31	284,5	17	65	40	
1	2010	34	98,35	19	70	30	
2	2010	30	132,23	20	70	60	
3	2010	28	44,63	14	55	70	
4	2010	34	69,3	19	65	45	
5	2010	38	62,48	19	55	65	
6	2010	32	42,8	19	50	30	
7	2010	31	114,65	15	70	45	
8	2010	28	45,55	15	45	40	
9	2010	30	37,23	18	45	55	
10	2010	31	56,6	15	60	35	
11	2010	30	29,9	3	60	60	

Abbildung 10: Eingangsdaten (Auszug) VOR Normalisierung

	0	1	2	3	4	5	6	7
0	0,4	0,2	0,2568370987	0,56	0,4042553191	0,2545454545	0,6785714286	0,5098039216
1	0,4	0,2	0,6582299983	0,4	0,3404255319	0,3272727273	0,2321428571	0,0980392157
2	0,2	0,5	0,1253609648	0,68	0,9361702128	0,2363636364	0,9107142857	0,4901960784
3	0,4	0,5	0,0202989638	0,6	0,7659574468	0,4545454545	0,8214285714	0,8235294118
4	0,4	0,4	0,1396806523	0,64	0,4042553191	0,3636363636	0,625	0,7843137255
5	0,4	0,35	0,0529131986	0,6	0,7872340426	0,6363636364	0,6071428571	0,2549019608
6	0,6	0,45	0,2073212162	0,44	0,7021276596	0,6	0,5535714286	0,5490196078
7	0,4	0,4	0,1632410396	0,32	0,5744680851	0,6	0,5535714286	0,568627451
8	0,2	1	0,1736028537	1	0,7234042553	0,4181818182	0,7857142857	0,1960784314
9	0,4	0,5	0,0202989638	0,6	0,7659574468	0,4545454545	0,8214285714	0,8235294118
10	0	0,2	0,0421776796	0,44	0,5106382979	0,8727272727	0,7857142857	0,3725490196
11	0	0,5	0,1334295906	0,64	0,829787234	0,1454545455	0,875	0,568627451

Abbildung 11: Eingangsdaten (Auszug) NACH Normalisierung

1. FC Köln_H	1. FC Nürnberg_H	1. FSV Mainz 05_H	Bayer 04 Leverkusen_H	Borussia Dortmund_H	Borussia Mönchengladbach_H
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	1
1	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	1	0
0	0	1	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0

Abbildung 12: Ergebnis des One-Hot-Encoding (Auszug)

7.4 Anhang D: Code für Training des Neuronales Netzes

Funktion zum Aufruf und Training des Neuronalen Netzes:

```
def neuronal_network(self, epochs, patience_early_stopping, threshold_for_early_stopping):

    #Funktion für das Ansprechen und Ausführen des Neuronalen Netzes mittels Pytorch


    #Standardausgabe für Pytorch, auf welcher processing unit gerechnet wird
    #In meinem Falle nur CPU möglich
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    print('This Computation is running on {}'.format(device))


    #Initialisierung des Neuronalen Netzwerks
    nn_model = NN_Model()
    #Als Fehlerfunktion wird CrossEntropyLoss verwendet
    loss_func = torch.nn.CrossEntropyLoss()
```

```
#Als Optimizer der Adam-Optimizer mit einer learing rate von 0.001
optimizer = torch.optim.Adam(nn_model.parameters(), lr=0.001)

#Leere Arrays für das Speichern von Fehler-/Akkuranzdaten über die Epochen hinweg
epoch_errors = []
epoch_train_accuracy = []
epoch_test_accuracy = []

#Initialisierung des Early Stopping
early_stopping = EarlyStopping(patience=patience_early_stopping)

#Umsetzen der Trainings- und Testdaten in die benötigten Tensoren-Formate
X_Train = torch.from_numpy(self.X_train).float()
y_Train = torch.tensor(self.y_train, dtype=torch.long)
X_Test = torch.from_numpy(self.X_test).float()
y_Test = torch.from_numpy(np.array(self.y_test)).long()

#Trainieren des Neuronalen Netzwerks; maximale Anzahl der Epochen als Funktionsparameter übergeben
for epoch in range(epochs):

  #Vorbereiten der Ergebnisse des Neuronalen Netzwerkes
  #LogSoftmax explizit hier, da diese in der Fehlerfunktion (CrossEntropyLoss) automatisch
  #angewandt wird!
  log_sm = torch.nn.LogSoftmax(dim=1)
  train_nn_model = log_sm(nn_model(X_Train))
  test_nn_model = log_sm(nn_model(X_Test))

  #Erstellen von leerem Array für das Speichern der einzelnen vom Modell berechneten Ergebnisse
  #Zusätzlich noch ein Zähler zum Aufsummieren der korrekt vorhergesagten Ergebnisse, mit 0 initialisiert
  train_pred_ergebnis = []
  train_running_correct = 0

  test_pred_ergebnis = []
  test_running_correct = 0

  #Autograd ausschalten für das Berechnen der Ergebnisse zu Validierungszwecken
  with torch.no_grad():
  #Trainings-Akkuranz
  # Leeren array füllen mit Ergebnissen aus Ergebnis-Tensor
  # Hierbei werden die probalistischen Werte verglichen und das wahrscheinlichste Ergebnis übergeben
  # als 0 - Heimsieg, 1 - Unentschieden, 2 - Auswärtssieg
    for i in range(train_nn_model.shape[0]):
```

```
        ergebnis = 0 if (train_nn_model[i][0] > train_nn_model[i][1] and train_nn_model[i][0] > train_nn_model[i][2])
else 1 if (train_nn_model[i][1] > train_nn_model[i][0] and train_nn_model[i][1] > train_nn_model[i][2]) else 2
        train_pred_ergebnis.append(ergebnis)
    #Test-Akkuranz
    # Leeren array füllen mit Ergebnissen aus Ergebnis-Tensor
    # Hierbei werden die probalistischen Werte verglichen und das wahrscheinlichste Ergebnis übergeben
    # als 0 - Heimsieg, 1 - Unentschieden, 2 - Auswärtssieg
      for i in range(test_nn_model.shape[0]):
        ergebnis = 0 if (test_nn_model[i][0] > test_nn_model[i][1] and test_nn_model[i][0] > test_nn_model[i][2]) els
e 1 if (test_nn_model[i][1] > test_nn_model[i][0] and test_nn_model[i][1] > test_nn_model[i][2]) else 2
        test_pred_ergebnis.append(ergebnis)

    #Arrays in tensor umwandeln
    train_pred_tensor = torch.tensor(train_pred_ergebnis, dtype=torch.float)
    test_pred_tensor = torch.tensor(test_pred_ergebnis, dtype=torch.float)

    #Die korrekten Ergebnisse aus dem Traininsdatensatz werden aufsummiert und
    #daraus die Akkuranz dieser Epoche berechnet und dem Array epoch_train_accuracy für spätere Auswertung übergeben
    train_running_correct += (train_pred_tensor == y_Train).sum().item()
    train_accuracy = train_running_correct*100./y_Train.shape[0]
    epoch_train_accuracy.append(train_accuracy)

    #Die korrekten Ergebnisse aus dem Testdatensatz werden aufsummiert und
    #daraus die Akkuranz dieser Epoche berechnet und dem Array epoch_test_accuracy für spätere Auswertung übergeben
    test_running_correct += (test_pred_tensor == y_Test).sum().item()
    test_accuracy = test_running_correct*100./y_Test.shape[0]
    epoch_test_accuracy.append(test_accuracy)

    #-----------------------------------------------------------------------------------------
    #Hier werden nun die entscheidenden Schritte zum Trainineren des NN Modells durchgeführt
    #-----------------------------------------------------------------------------------------
    error = loss_func(nn_model(X_Train),y_Train)
    optimizer.zero_grad()
    error.backward()
    epoch_errors.append(error.item())
    optimizer.step()
    #-----------------------------------------------------------------------------------------

    #Debug-Print Ausgabe der Epoche mit Akkuranzen
    print("Epoche: {}/{} mit Train-Akkuranz: {} und Test-Akkuranz: {}".format(epoch, epochs, train_accuracy, test_ac
curacy))

    #-----------------------------
```

```
        #Early Stopping
        #-----------------------------
        #Loss für Testdaten berechnen
        error_Test = loss_func(nn_model(X_Test),y_Test)

        #Aufruf der Early Stopping Funktion
        # Die Fehlerfunkion der Testdaten dient hier als zentrales Kriterium:
        # Sinkt diese mit der Rate "delta" eine bestimmte Anzahl Schritte "patience"
        # hintereinander NICHT MEHR, wird gestoppt.
        # Zusätzlich wird ein Threshold mit angegeben, sodass erst ab einer bestimmten erreichten
        # Akkuranz das Early Stopping aktiviert wird.
        early_stopping(error_Test, nn_model, train_accuracy > threshold_for_early_stopping)
        #Sollte ein Early Stop erreicht sein, wird das Durchlaufen der Epochen unterbrochen
        if early_stopping.early_stop:
          print("Early stopping")
          break
        #-----------------------------

    #Debug-Print finales Loss-Ergebnis
    #print('Loss nach {} Epochen: {}'.format(epoch+1,error.item()))

    #Übergabe der Ergebnisdaten and den zentralen Ergebnis-Array
    self.ergebnis.append([nn_model.__class__.__name__, test_accuracy/100, nn_model])

    #Rückgabewerte für weitere Verwendung (Ausgabe, Test) im Hauptprogramm
    return self.ergebnis, epoch_errors, epoch_train_accuracy, epoch_test_accuracy, test_pred_tensor
```