DAV

DEUTSCHE
AKTUARVEREINIGUNG e.V.

Praktische Prüfung im Vertiefungswissen

Actuarial Data Science Completion

gemäß Prüfungsordnung 4
der Deutschen Aktuarvereinigung e. V.

Zeitraum: 12.09.2022 – 12.10.2022

Hinweise:

- Die Gesamtpunktzahl beträgt 180 Punkte. Die Prüfung ist bestanden, wenn mindestens 90 Punkte erreicht werden.
- Bitte prüfen Sie die Ihnen vorliegende Prüfungsunterlagen auf Vollständigkeit. Die Prüfung besteht aus den Blöcken A und B und beinhaltet insgesamt 7 Seiten. Zusätzliche Materialen (Datensätze, etc.) sollten Sie mit der Prüfung erhalten haben. Auf diese wird in den Blöcken eingegangen.
- Für jeden Block ist ein PDF-Dokument einzureichen. Falls ein Notebook zu erstellen ist, dann ist dieses zusätzlich einzureichen. Der Output erstellter Notebooks muss mindestens im eingereichten PDF-Dokument, falls möglich auch im Notebook sichtbar sein. Aufgabenteile mit erforderlichem, aber fehlendem Output werden mit 0 Punkten bewertet.
- Für jeden Block sind die Aufgaben in der vorgegebenen Reihenfolge zu bearbeiten. Die Lösung muss direkt nach der Aufgabenstellung in einer strukturierten und übersichtlichen Weise dargestellt werden. Nichtbeachtung dieser Vorgabe führt zu Punkteabzug.
- Mit der Einreichung eines Ergebnisnotebooks erklärt die zu prüfende Person, dass der vorliegende Ergebnisbericht inklusive Code- und Beschreibungsanteilen eigenständig erzeugt worden ist. Verstöße gegen diese Grundlagen können vom Prüfungsausschuss sanktioniert werden und zum Ausschluss von der Prüfung führen.

Mitglieder der Prüfungskommission:

Renate Ernst, Daniela Giesinger, Dr. René Külheim, Friedrich Loser, Prof. Dr. Fabian Transchel, Prof. Dr. Christian Weiß

Block A (Erklärbarkeit) [75 Punkte]

Hinweise:

- Für diesen Block ist ein Notebook (R-Markdown oder Jupyter Notebook) mit R oder Python zu erstellen.
- Benötigte Materialien: Teil A.csv

Aufgabe A1 Explorative Datenanalyse [Lernziel 6.1.1]

[5 Punkte]

Lesen Sie die bereitgestellten Daten aus Teil A.csv ein und führen Sie eine explorative Datenanalyse durch. Welche Merkmale liegen vor und wie sind diese verteilt? Führen Sie eine Kerndichteschätzung durch und fertigen Sie einen aussagekräftigen Scatterplot der Merkmale X0 und X1 an.

Aufgabe A2 Basismodell [Lernziel 6.1.1]

[20 Punkte]

Erzeugen Sie ein klassifizierendes Künstliches Neuronales Netz, das die vorgegebenen Klassen zuordnen lernt. Wählen Sie eine geeignete Netztopologie und prüfen Sie die Performance anhand der Betrachtung von Accuracy, Confusion Matrix und One-vs-Rest-ROC-Kurven. Visualisieren Sie die zugeordneten Klassen bezüglich der Labels. Markieren Sie spezifisch falsch klassifizierte Beispiele und erläutern Sie, ob diese Beispiele strukturelle Eigenschaften verbinden.

Aufgabe A3 Maximum Activation Analysis [Lernziele 5.3.5, 5.3.6]

[20 Punkte]

Es soll eine „Maximum Activation Analysis" durchgeführt werden.

Visualisieren Sie die Aktivierungswerte der einzelnen Knoten des Netzes:

- für die Datensätze mit den Indizes 0, 1, 500
- im Mittel
- bezogen auf die vier Quadranten, also nach x1- / x2-Werten (bzgl. der Achsen)
- differenziert nach Label des Inputs, also auf Klassenebene

Dabei sollen die jeweiligen Maximum Activation Plots die Struktur des Netzes geeignet wiedergeben. Welche Einsichten lassen sich hieraus gewinnen?

Aufgabe A4 Partial Dependence [Lernziele 5.3.5, 5.3.6]

[15 Punkte]

Ein Partial Dependence Plot soll erzeugt werden. Wählen Sie geeignete Darstellungsarten für die Merkmale X1 und X2 und erzeugen Sie auch 2D-Plots sowie 3D-Plots für beide Merkmale gemeinsam. Diskutieren Sie, ob die 2D- oder 3D-Darstellung für zwei Merkmale informativer ist.

Sind PDPs geeignet, die Wirkungsweise des erzeugten Classifiers zu illustrieren? Welche Schwierigkeiten zeigen sich dabei? Inwiefern kann ein PDP dabei helfen zu entscheiden, ob ein Modell einfacher oder komplexer werden muss?

Aufgabe A5 LIME [Lernziele 5.3.5, 5.3.6]

[5 Punkte]

Erstellen Sie LIME-Erklärungen für die Beobachtungen 0, 1 und 500 und erläutern Sie, inwiefern der LIME-Ansatz hier zum Verständnis des Modells beiträgt.

Aufgabe A6 Shapley-Values [Lernziele 5.3.5, 5.3.6]

[10 Punkte]

Führen Sie eine Erklärbarkeitsuntersuchung mit Hilfe von Shapley-Values durch. Erzeugen Sie summary-, dependence- und force-Plots. Ordnen Sie die Eignung der Methode zur Erlangung von Verständnis in Bezug auf das Basismodell ein, auch in Abgrenzung zu den vorigen Aufgaben (PDP, LIME).

Block B (Textklassifikation) [105 Punkte]

Hinweise:

- Für diesen Block ist ein Notebook (R-Markdown oder Jupyter Notebook) mit R oder Python zu erstellen.
- Benötigte Materialien: Schadenbeschreibung_Sparte-de.csv

Die Datenbasis Schadenbeschreibung_Sparte-de.csv enthält zwei durch Semikolon getrennte Merkmale:

1. 'Description': kurze englischsprachige Schadenbeschreibung.

2. 'line': Zuordnung zu einer Sparte/Risikoart bzw. Sonstige (Misc).

Aufgabenspezifische bzw. technische Vorbemerkungen:

Als Business Case soll auf Basis der Schadenbeschreibungen mittels Textmining eine automatische Spartenzuordnung entwickelt werden.

- Alternativ können in Aufgabe B7 anstelle der LSTM-Layer auch GRU-Layer, dann aber durchgängig, verwendet werden.

- Bei Verwendung mehrerer, insbesondere ähnlicher Programmbibliotheken können "name space"-Konflikte auftreten und müssen geeignet beseitigt werden.

- Bei Verwendung insbesondere älterer Code-Vorlagen aus öffentlichen Quellen können aufgrund von Versionsänderungen Anpassungen erforderlich sein (z.B. Unterschiede in TensorFlow 1 vs. 2).

- Zur Orientierung: Die gesamte Laufzeit des zur Aufgabe entwickelten Lösungsvorschlages beträgt auf einem betagten Notebook weniger als 10 Minuten. In diesem Laufzeitbudget ist eine Prognosegenauigkeit von (ggf. knapp) 90% erreichbar. Das PDF-Dokument ist einschließlich Output kleiner als 1 MB.

Aufgabe B1 Daten einlesen und visualisieren [Lernziel 6.1]

[5 Punkte]

Lesen Sie die Datei Schadenbeschreibung_Sparte-de.csv ein und vergewissern Sie sich, dass alle Datensätze korrekt verarbeitet wurden. Erstellen Sie eine Häufigkeitstabelle der Sparten (line) und zeigen Sie diese graphisch an. Geben Sie zusätzlich die drei längsten Beschreibungen ("Description", gemessen an der Anzahl Zeichen) aus.

Aufgabe B2 Schadenbeschreibungen aufbereiten [Lernziel 5.1]

[15 Punkte]

Erläutern Sie kurz die Textmining-Schritte "Stoppwörter entfernen" und Lemmatisierung und nennen Sie zu beiden Begriffen ein kurzes Beispiel.

Entfernen Sie aus dem Feld "Description" typische englischsprachige Stopwords und führen Sie eine Lemmatisierung durch. Zeigen Sie für eine kleine Auswahl an Datensätzen die ursprünglichen und die bereinigten Texte an. Ermitteln Sie die Häufigkeit der verbleibenden Wörter und zeigen Sie die Verteilung für alle Wörter, die mindestens 250-mal vorkommen, in einem Balkendiagramm absteigend sortiert an.

Das mit Abstand häufigste Wort ist "damage" (nach Lemmatisierung). Ermitteln Sie je Sparte (line) den Anteil der Schadenbeschreibungen, bei denen das Wort "damage" mindestens einmal vorkommt. Interpretieren Sie das Ergebnis und wägen Sie ab, ob "damage" als Stopword entfernt werden sollte.

Aufgabe B3 Trainings- und Testdaten erzeugen, Tokenisierung vornehmen und überprüfen [Lernziel 5.1]

[15 Punkte]

Teilen Sie die Datensätze in 75% Trainingsdaten und 25% Testdaten über eine reproduzierbare Zufallsauswahl auf. Führen Sie an den bereinigten Schadenbeschreibungen die Schritte Tokenisierung, Sequenzbildung und "Padding" durch. Dabei soll die Betrachtung auf die häufigsten 100 Worte sowie Sequenzen der Länge 10 begrenzt werden. Die Datenstrukturen sind so aufzubereiten, dass mit Keras eine multinominale Klassifikation (der Sparten) mit einem Word-Embedding (der Schadenbeschreibungen) in einem neuronalen Netz berechnet werden kann. Die Datenaufbereitung soll sowohl für die Trainings- als auch für die Testdaten durchgeführt werden. Zeigen Sie für jeweils zwei geeignete Trainings- und Test-

datensätze die ursprüngliche Schadenbeschreibung, die bereinige Schadenbeschreibung und die Zahlendarstellung als Token. Zeigen Sie an Beispielen, dass die Token in den Trainings- und Testdaten die gleiche Wortbedeutung haben.

Aufgabe B4 Spartenvorhersage mit einem Neuronalen Netz und Word Embedding [Lernziel 5.1]

[20 Punkte]

Auf Basis der Schadenbeschreibungen soll die Sparte vorhergesagt werden (multinominale Klassifikation). Erstellen Sie hierfür mit Keras ein vorwärts gerichtetes, voll verknüpftes neuronales Netz (FC) mit einem 32-dimensionalen "Embedding Layer" und einem "Hidden Layer" mit 32 Neuronen. Zeigen Sie die Modellgröße an ("summary"). Trainieren Sie das Modell (max. 20 epochs) mit Verwendung einer internen Validierungsstichprobe von 25%, zeigen Sie die Entwicklung von "Loss" und "Accuracy" über die Trainingsdurchläufe (epochs) an und bewerten Sie die Konvergenz. Erstellen Sie eine Sparten-Vorhersage anhand der Testdaten. Ermitteln und zeigen Sie die resultierende "Confusion Matrix" sowie Modellgenauigkeit und interpretieren Sie beides.

Aufgabe B5 Word Embedding analysieren [Lernziel 5.1]

[15 Punkte]

Erstellen Sie mit Hilfe der Dimensionreduktionsverfahren UMAP oder t-SNE eine zweidimensionale Darstellung des hochdimensionalen Embeddings aus Aufgabe B4. Beschreiben und begründen Sie Ihr ausgewähltes Verfahren kurz und erläutern Sie die verwendeten Parameter. Erstellen Sie eine zweidimensionale graphische Darstellung des Ergebnisses der Dimensionsreduktion. Heben Sie dabei Worte, die auch Spartennamen sind, farblich hervor. Kommentieren und bewerten Sie die dargestellte Wortverteilung im zweidimensionalen Raum. Berechnen Sie zum Vergleich die ähnlichsten Worte zu "water" sowie "storm" gemäß "Cosine Similarity" der entsprechenden Wortvektoren und plausibilisieren Sie damit die graphische Darstellung.

Aufgabe B6 Kreuzvalidierung und Embedding-Größe [Lernziel 6.1]

[15 Punkte]

Führen Sie auf Basis des in Aufgabe B4 entwickelten Vorhersagemodells für Embeddings mit 2, 4, 8, 16, 32 und 64 Dimensionen eine einheitliche, selbst programmierte fünffache Kreuzvalidierung für die sechs Modelle durch. Berechnen Sie die jeweilige Modellgenauigkeit (Accuracy) an der entsprechenden Validierungsstichprobe und ermitteln Sie über die fünf Validierungsstichproben den Mittelwert

der Modellgenauigkeit für jedes der sechs Modelle. Visualisieren Sie die Modellgenauigkeit mittels Boxplots. Zeigen Sie dazu auf der x-Achse die Embedding-Größe und auf der y-Achse die Modellgenauigkeit als Boxplot an und interpretieren Sie das Ergebnis.

Aufgabe B7 Spezielle neuronale Netze kreuzvalidiert anwenden [Lernziel 4.2]

[20 Punkte]

a) Ziel dieser Aufgabe ist festzustellen, ob durch rekurrente neuronale Netze (RNN) und andere tiefe spezielle neuronale Netze wie Faltungsnetze (ConvNet) das Prognoseergebnis verbessert werden kann. Erläutern Sie, weshalb hier RNNs und ConvNets bzw. alternativ zu ConvNets ein von Ihnen bevorzugtes neueres Deep-Learning-Verfahren zu einer Prognoseverbesserung führen könnten.

b) Zur technischen Umsetzung können in Aufgabe B6 entwickelte Programmteile verwendet bzw. adaptiert werden. Die fünffache Kreuzvalidierung soll beibehalten werden. Es soll wie bisher Keras genutzt und in jedem neuronalen Netz eingangs ein Word Embedding verwendet werden, nun einheitlich mit 32 Dimensionen. Anstelle mehrerer neuronalen Netze mit unterschiedlichen Embedding-Größen (Aufgabe B6) sollen nun mehrere RNNs mit unterschiedlicher Anzahl an "units" (Größe des Ausgaberaums) berechnet werden. Konkret soll dazu im Vergleich zu Aufgabe B6 (mindestens) die Hidden Layer entfernt und durch LSTM-Layer mit 8, 16 bzw. 32 units (drei Modelle) ersetzt werden. Zusätzlich soll ein anderes tiefes spezielles neuronales Netz, beispielsweise mit einer eindimensionalen Faltungsschicht (conv-1d-layer) geeignet mit in die Betrachtung einbezogen werden. Analog Aufgabe B6 soll die jeweilige Modellgenauigkeit (Accuracy) an der entsprechenden Validierungsstichprobe berechnet werden und über die fünf Validierungsstichproben den Mittelwert der Modellgenauigkeit für jedes der vier Modelle ermittelt werden.

c) Visualisieren Sie die Prognosegenauigkeit der vier neuen Modelle analog Aufgabe B6 mittels Boxplots. Nehmen Sie zusätzlich die entsprechenden Werte des (voll verknüpften vorwärts gerichteten) FC-Modells mit 32-dimensionalem Embedding aus Aufgabe B6 mit in den Boxplot auf und interpretieren Sie die Genauigkeit samt Schwankungen.

d) Berechnen Sie für die jeweilige Validierungsstichprobe die Differenz der Prognosegenauigkeit der einzelnen vier neuen Modelle mit dem oben genannten FC-Modell und visualisieren Sie die Differenzen in einen weiteren Boxplot für die vier neuen Modelle. Erläutern Sie, was durch die Differenzbildung erreicht wurde und interpretieren sie das Ergebnis.

Lösungsvorschlag zur Prüfungsaufgabe Completion 2022

Teil A: Erklärbarkeit [75 Punkte]

Benötigte Bibliotheken einbinden

In [1]:
```
import pandas as pd
import numpy as np

from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc
from sklearn.inspection import PartialDependenceDisplay
from sklearn.inspection import partial_dependence

import tensorflow as tf
from tensorflow import keras
from keras import models
from keras.wrappers.scikit_learn import KerasClassifier
tf.random.set_seed(1111)

import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
import mpl_toolkits.mplot3d

import lime
import lime.lime_tabular
import lime.lime_image

import shap

import warnings
warnings.filterwarnings("ignore", category=UserWarning)
```

Aufgabe 1: Explorative Datenanalyse (5 Punkte, Lernziel 6.1.1)

Lesen Sie die bereitgestellten Daten aus `Teil A.csv` ein und führen Sie eine explorative Datenanalyse durch. Welche Merkmale liegen vor und wie sind diese verteilt? Führen Sie eine Kerndichteschätzung durch und fertigen Sie einen aussagekräftigen Scatterplot der Merkmale `X0` und `X1` an.

In [2]:
```
df = pd.read_csv("Teil A.csv")
df.head()
```

Out[2]:

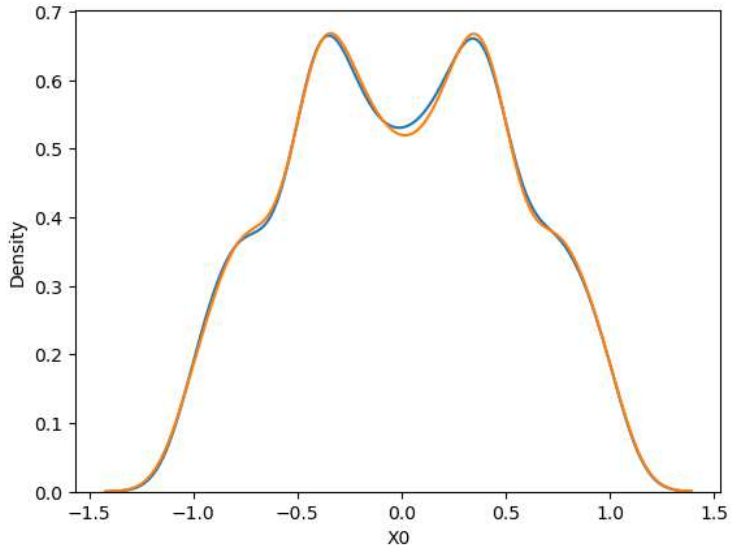
	Unnamed: 0	X0	X1	label
0	0	-0.687698	-0.452663	1
1	1	-0.114451	-0.388861	3
2	2	-0.619078	0.402220	1
3	3	-1.003385	0.071266	0
4	4	0.716414	0.674429	0

In [3]:
```
df.describe()
```

Out[3]:

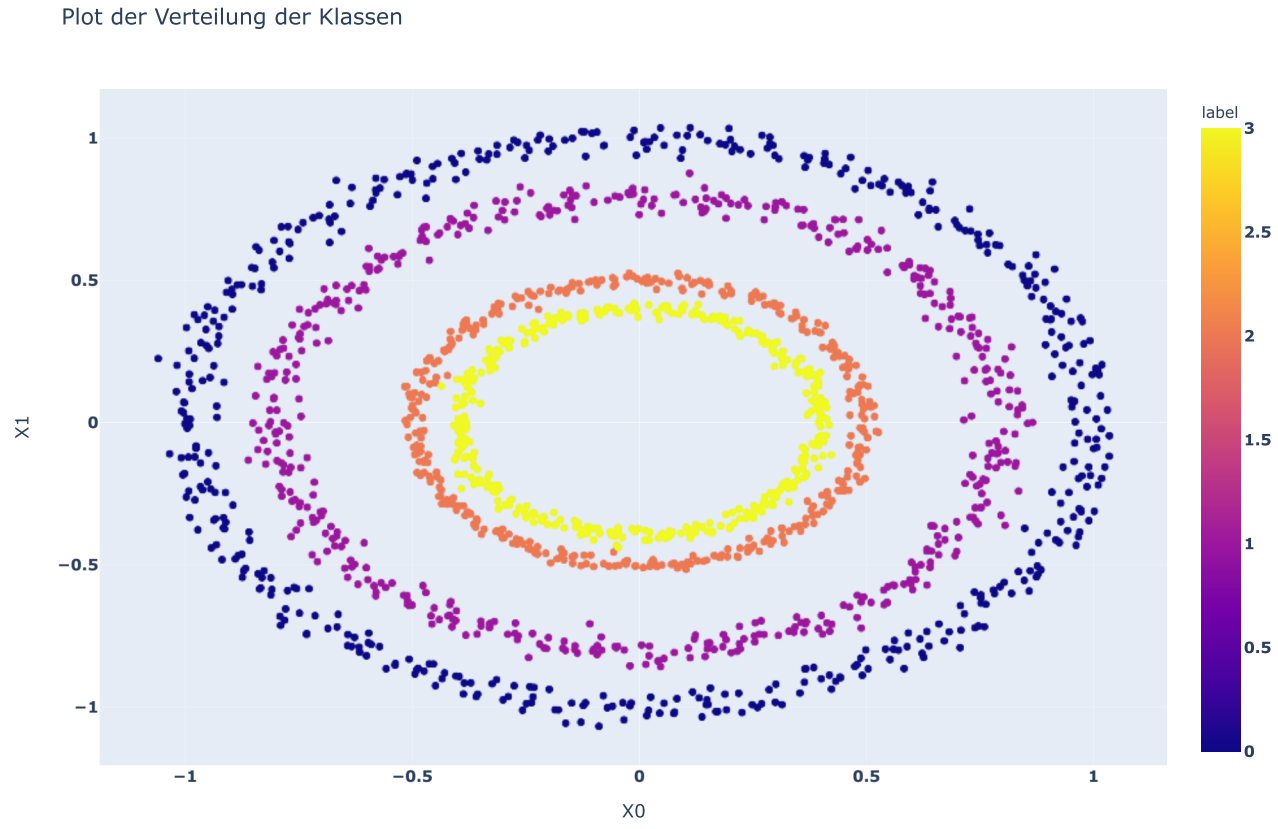
	Unnamed: 0	X0	X1	label
count	1750.000000	1750.000000	1750.000000	1750.000000
mean	874.500000	-0.000237	-0.000708	1.357143
std	505.325802	0.526980	0.528226	1.109187
min	0.000000	-1.059502	-1.067768	0.000000
25%	437.250000	-0.401385	-0.398716	0.000000
50%	874.500000	0.003570	-0.001578	1.000000
75%	1311.750000	0.402071	0.401975	2.000000
max	1749.000000	1.033712	1.036004	3.000000

In [4]:
```
sns.kdeplot(df.X0)
sns.kdeplot(df.X1)
plt.show()
```



Die vorgenommene Kerndichteschätzung legt nahe, dass beide Merkmale nicht nur sehr ähnlich, sondern auch eher ungewöhnlich verteilt sind: Es liegt für `X0` und `X1` eine Art Bimodale Verteilung vor.

In [5]:
```
px.scatter(df,x='X0',y='X1',color='label',width=1000,height=700,title="Plot der Verteilung der Klassen")
```



Der Scatterplot verdeutlicht, dass die Beispiele in vier ringartigen Strukturen angeordnet sind. Diese sind jeweils einer der vier in `label` codierten Klassen zugeordnet. Hierbei ist zu bemerken, dass die Abständen zwischen den Ringstrukturen nicht gleichmäßig sind.

Aufgabe 2: Basismodell (20 Punkte, Lernziel 6.1.1)

Erzeugen Sie ein klassifizierendes Künstliches Neuronales Netz, das die vorgegebenen Klassen zuordnen lernt. Wählen Sie eine geeignete Netztopologie und prüfen Sie die Performance anhand der Betrachtung von Accuracy, Confusion Matrix und *One-vs-Rest*-ROC-Kurven. Visualisieren Sie die zugeordneten Klassen bezüglich der Labels. Markieren Sie spezifisch falsch klassifizierte Beispiele und erläutern Sie, ob diese Beispiele strukturelle Eigenschaften verbinden.

In [6]:
```
X = np.array(df.drop(['Unnamed: 0','label'],axis=1))
y = df.label
enc = OneHotEncoder(handle_unknown='ignore',sparse=False)
```

```
enc.fit(np.array(y).reshape(-1,1))
y = enc.transform(np.array(y).reshape(-1,1))
```

In [7]:
```
model = keras.Sequential([
        keras.layers.Dense(6, activation='relu', input_shape=[2], kernel_initializer='glorot_uniform'),
        keras.layers.Dense(6, activation='relu'),
        keras.layers.Dense(4, activation='softmax')
        ])

model.compile(loss= tf.keras.losses.categorical_crossentropy
              ,optimizer = tf.keras.optimizers.Adam(learning_rate = 0.01)
              ,metrics=['accuracy'])

model.fit(X, y, epochs = 25, verbose = 1)
```

```
Epoch 1/25
55/55 [==============================] - 0s 575us/step - loss: 1.3205 - accuracy: 0.2966
Epoch 2/25
55/55 [==============================] - 0s 537us/step - loss: 1.0777 - accuracy: 0.4571
Epoch 3/25
55/55 [==============================] - 0s 537us/step - loss: 0.8112 - accuracy: 0.7280
Epoch 4/25
55/55 [==============================] - 0s 555us/step - loss: 0.6705 - accuracy: 0.8851
Epoch 5/25
55/55 [==============================] - 0s 537us/step - loss: 0.5500 - accuracy: 0.9314
Epoch 6/25
55/55 [==============================] - 0s 500us/step - loss: 0.4499 - accuracy: 0.9451
Epoch 7/25
55/55 [==============================] - 0s 500us/step - loss: 0.3560 - accuracy: 0.9623
Epoch 8/25
55/55 [==============================] - 0s 574us/step - loss: 0.2867 - accuracy: 0.9714
Epoch 9/25
55/55 [==============================] - 0s 574us/step - loss: 0.2307 - accuracy: 0.9771
Epoch 10/25
55/55 [==============================] - 0s 500us/step - loss: 0.1983 - accuracy: 0.9754
Epoch 11/25
55/55 [==============================] - 0s 537us/step - loss: 0.1650 - accuracy: 0.9766
Epoch 12/25
55/55 [==============================] - 0s 556us/step - loss: 0.1443 - accuracy: 0.9811
Epoch 13/25
55/55 [==============================] - 0s 500us/step - loss: 0.1327 - accuracy: 0.9766
Epoch 14/25
55/55 [==============================] - 0s 518us/step - loss: 0.1142 - accuracy: 0.9846
Epoch 15/25
55/55 [==============================] - 0s 537us/step - loss: 0.1058 - accuracy: 0.9811
Epoch 16/25
55/55 [==============================] - 0s 463us/step - loss: 0.0946 - accuracy: 0.9840
Epoch 17/25
55/55 [==============================] - 0s 633us/step - loss: 0.0912 - accuracy: 0.9811
Epoch 18/25
55/55 [==============================] - 0s 520us/step - loss: 0.0894 - accuracy: 0.9806
Epoch 19/25
55/55 [==============================] - 0s 537us/step - loss: 0.0803 - accuracy: 0.9823
Epoch 20/25
55/55 [==============================] - 0s 519us/step - loss: 0.0727 - accuracy: 0.9846
Epoch 21/25
55/55 [==============================] - 0s 500us/step - loss: 0.0664 - accuracy: 0.9874
Epoch 22/25
55/55 [==============================] - 0s 537us/step - loss: 0.0704 - accuracy: 0.9823
Epoch 23/25
55/55 [==============================] - 0s 499us/step - loss: 0.0628 - accuracy: 0.9886
Epoch 24/25
55/55 [==============================] - 0s 520us/step - loss: 0.0614 - accuracy: 0.9880
Epoch 25/25
55/55 [==============================] - 0s 574us/step - loss: 0.0620 - accuracy: 0.9840
```

Out[7]:
```
<tensorflow.python.keras.callbacks.History at 0x1e89fc4dbb0>
```

In [8]:
```
model.evaluate(X,y)
```

```
55/55 [==============================] - 0s 723us/step - loss: 0.0609 - accuracy: 0.9851
```

Out[8]:
```
[0.06087247654795647, 0.9851428866386414]
```

Im Folgenden wird eine simple Routine geschrieben, um die Softmax-Outputs zurück auf die Labels zu mappen. Anhand der Beobachtung einzelner Beispiele lässt sich schließen, dass der Wrapper korrekt funktioniert und wir mit der Betrachtung der Gütemaße beginnen können.

In [9]:
```
one_hot_predict = lambda x:np.argmax(np.round(model.predict(X)),axis=1)
predictions = one_hot_predict(X)
```

In [10]:
```
# Welche Klassen kommen vor?
print(np.unique(predictions))
```

```
[0 1 2 3]
```

In [11]:
```
# die in späteren Aufgabenteilen verwendeten Indizes
print(predictions[0])
print(predictions[1])
print(predictions[500])

# sonstige
print(predictions[3])
print(predictions[1500])
```

```
1
3
1
0
2
```

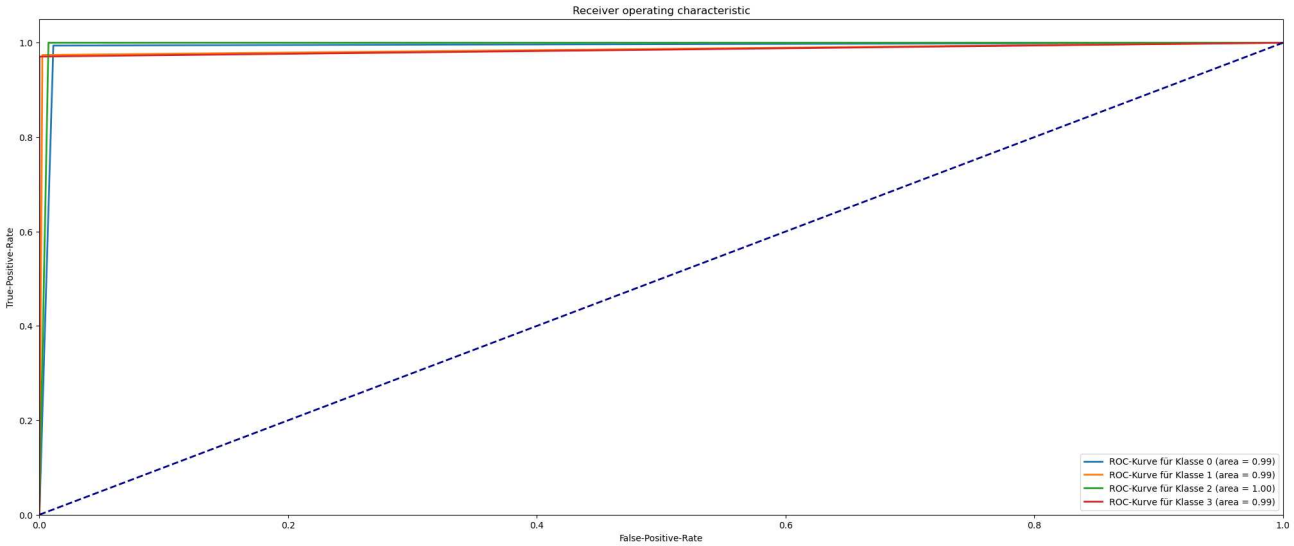
Aus dem Evaluationsbericht ist bereits erkennbar, dass die Accury jenseits von 98% liegt, was als akzeptables Modell gelten kann. Vor der Bearbeitung der weiteren Aufgaben muss geprüft werden, ob das Modell geeignet ist, den grundlegenden Zusammenhang zu approximieren. Hierzu wird die Confusion Matrix und die ROC-Curve ausgewertet.

In [12]:
```
y_true = np.argmax(y,axis=1)
y_pred = predictions
confusion_matrix(y_true,y_pred)
```

Out[12]:
```
array([[497,   3,   0,   0],
       [ 13, 487,   0,   0],
       [  0,   0, 375,   0],
       [  1,   0,  10, 364]], dtype=int64)
```

Um ROC-Kurven zu erzeugen muss zunächst aus dem encodierten Output die jeweilige One-vs-Rest-Verteilung bestimmt werden.

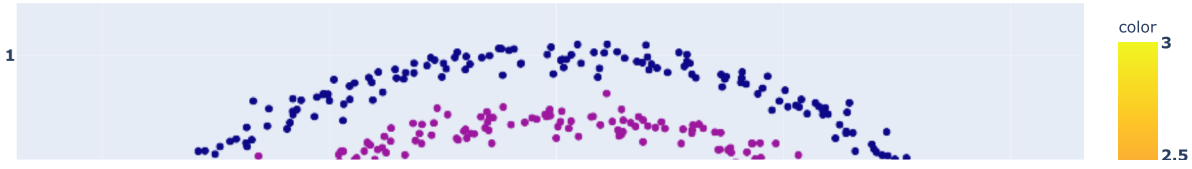
In [13]:
```
def class_mapper(pred,pos_class):
    if pred == pos_class:
        return 1
    else:
        return 0
    return 0

def ovr_encoder(pred,pos_class):
    return list(map(lambda x:class_mapper(x,pos_class),pred))

ovr_y_true = dict()
ovr_y_pred = dict()
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(0,4):
    ovr_y_true[i] = ovr_encoder(y_true,i)
    ovr_y_pred[i] = ovr_encoder(y_pred,i)
    fpr[i], tpr[i], _ = roc_curve(ovr_y_true[i], ovr_y_pred[i])
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.rcParams["figure.figsize"] = (25,10)
plt.figure()

lw = 2

for i in range(0,4):
    plt.plot(
        fpr[i],
        tpr[i],
        lw=lw,
        label="ROC-Kurve für Klasse "+str(i)+" (area = %0.2f)" % roc_auc[i],
    )
plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False-Positive-Rate")
plt.ylabel("True-Positive-Rate")
plt.title("Receiver operating characteristic")
plt.legend(loc="lower right")
plt.show()
```

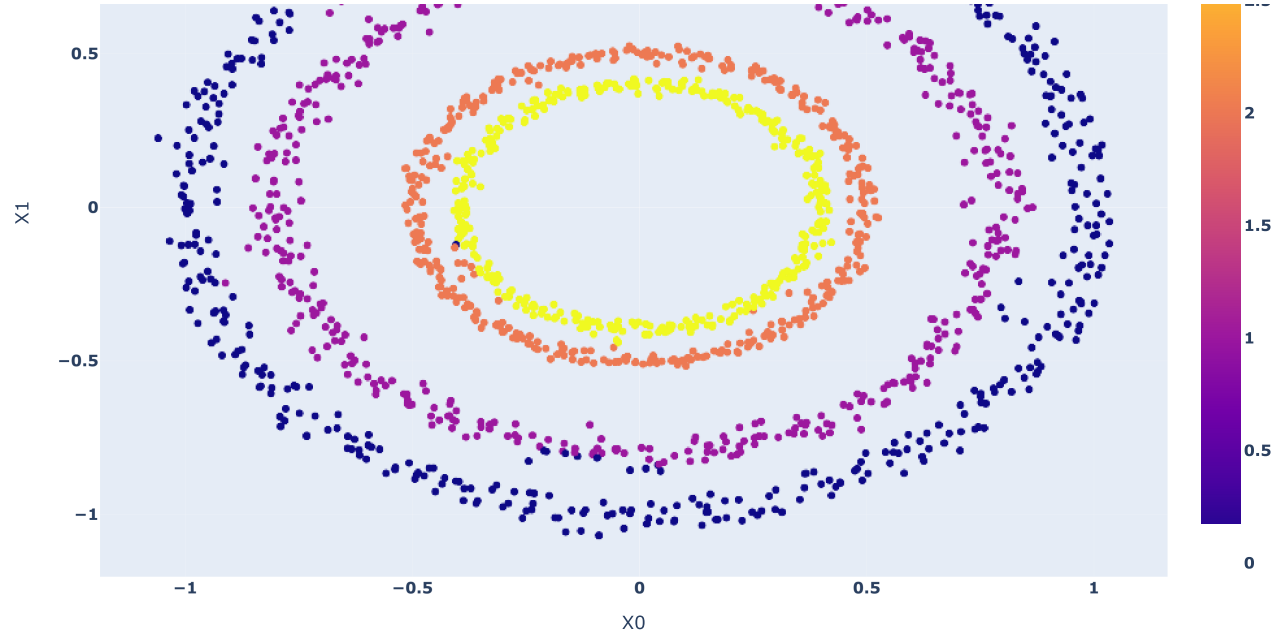


Erwartungsgemäß informiert uns auch die ROC-Darstellung für die One-vs-Rest-Darstellung darüber, dass das Modell sehr gut funktioniert. Die minimalen Unterschiede zwischen den Klassen sind auf einzelne falsch zugeordnete Beispiele zurückzuführen.

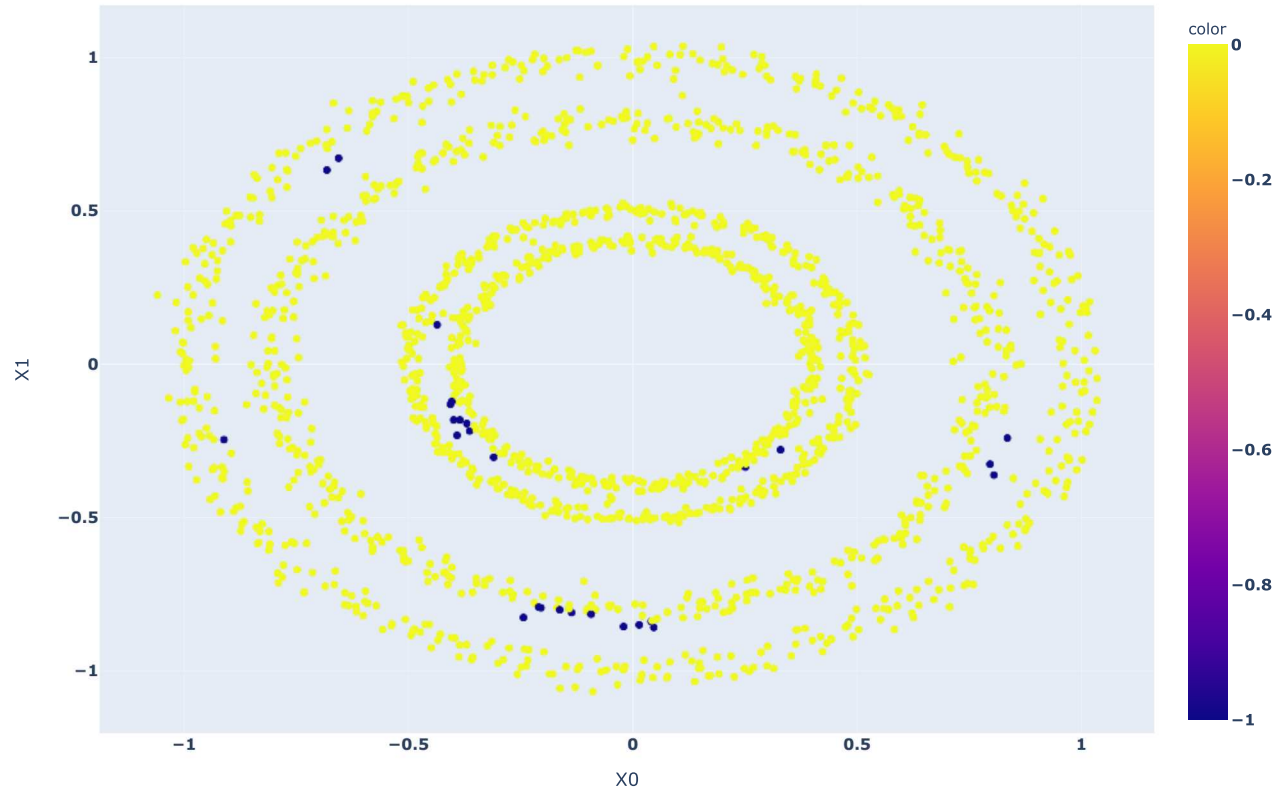
Wir schließen die Plausibilisierung damit ab, falsch klassifizierte Beispiele im Scatterplot zu markieren.

In [14]:
```
px.scatter(df,x='X0',y='X1', color = predictions,width=1000,height=700)
```





In [15]:
```
wrong_point_mask = -np.clip(0,1,np.abs(y_pred-y_true))
px.scatter(df,x='X0',y='X1', color = wrong_point_mask,width=1000,height=700)
```



Aus der Negativdarstellung der Clip-Maske erkennt man, dass die falsch klassifizierten Beispiele tendenziell in den Außenbereichen der einzelnen Ringe liegen, bzw. sogar eher *dazwischen*. Die Punkte sind bis auf einzelne Gegenbeispiele nachvollziehbar besonders schwierig zu lernen. Ein breiteres oder länger trainiertes Netz sollte prinzipiell in der Lage sein, diese noch korrekt zuzuordnen, allerdings ist fraglich, ob hierfür der Aufwand gerechtfertigt ist.

Aufgabe 3: Maximum Activation Analysis (20 Punkte, Lernziele 5.3.5, 5.3.6)

Es soll eine *Maximum Activation Analysis* durchgeführt werden.

Visualisieren Sie die Aktivierungswerte der einzelnen Knoten des Netzes:

* für die Datensätze mit den Indizes 0, 1, 500
* im Mittel
* bezogen auf die vier Quadranten, also nach x1- / x2-Werten (bzgl. der Achsen)
* differenziert nach Label des Inputs, also auf Klassenebene

Dabei sollen die jeweiligen Maximum Activation Plots die Struktur des Netzes geeignet wiedergeben. Welche Einsichten lassen sich hieraus gewinnen?

In [16]:
```
layer_outputs = [layer.output for layer in model.layers]

activation_model = models.Model(inputs=model.input, outputs=layer_outputs)

activations = activation_model.predict(X)

first_layer_activation = activations[0]
second_layer_activation = activations[1]
third_layer_activation = activations[2]
```

In [17]:
```
print(first_layer_activation[0])
print(first_layer_activation[1])
print(first_layer_activation[500])
print(second_layer_activation[0])
print(second_layer_activation[1])
print(second_layer_activation[500])
print(third_layer_activation[0])
print(third_layer_activation[1])
print(third_layer_activation[500])
```

```
[0.        0.        1.6067289 0.        0.        0.       ]
[0.18537232 0.         0.29389492 0.         0.         0.        ]
[0.        0.        0.        0.        0.726697  1.3314528]
[1.5938021  0.10562205 1.471306   0.         2.5139546  2.0238786 ]
[0.         0.         0.         0.         0.03772449 0.        ]
[1.914455   0.03473425 1.7441112  0.         3.2658157  2.8401763 ]
[3.8595196e-02 9.6076244e-01 6.4239569e-04 1.6179435e-13]
[3.8391116e-04 3.5562401e-03 2.0577464e-02 9.7548234e-01]
[5.1080422e-03 9.9440080e-01 4.9110613e-04 1.2055203e-18]
```

In [18]:
```
first_layer_coordinates = np.array(list(zip([1]*len(first_layer_activation[0]),list(range(len(first_layer_activation[0])))))
second_layer_coordinates = np.array(list(zip([2]*len(first_layer_activation[0]),list(range(len(second_layer_activation[0])))))
third_layer_coordinates = np.array(list(zip([3]*len(first_layer_activation[0]),list(range(1,1+len(third_layer_activation[0])))))
all_coordinates = np.array([*first_layer_coordinates,*second_layer_coordinates,*third_layer_coordinates])
all_coordinates
```

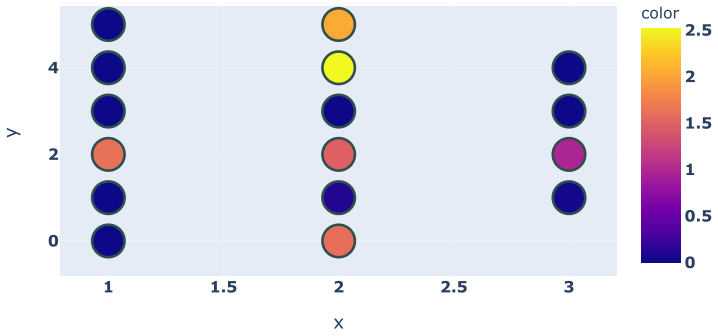
Out[18]:
```
array([[1, 0],
       [1, 1],
       [1, 2],
       [1, 3],
       [1, 4],
       [1, 5],
       [2, 0],
       [2, 1],
       [2, 2],
       [2, 3],
       [2, 4],
       [2, 5],
       [3, 1],
       [3, 2],
       [3, 3],
       [3, 4]])
```

Darstellung für die Aktivierung der Beispiele mit Index *0, 1* und *500*

In [19]:
```
def show_activation(all_activations, my_title):
    fig = px.scatter(all_coordinates
                     ,x=all_coordinates[:,0]
                     ,y=all_coordinates[:,1]
                     ,color=all_activations
                     ,width=600
                     ,height=400
                     ,title=my_title)

    fig.update_traces(marker=dict(size=25,
                                  line=dict(width=2,
                                            color='DarkSlateGrey')),
                      selector=dict(mode='markers'))

    fig.show()
```

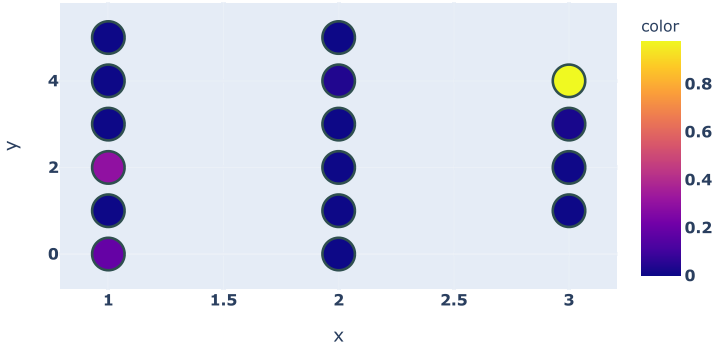
In [20]:
```
def get_all_activations_index(observation_index):
    all_activations = np.array([*first_layer_activation[observation_index]
                                ,*second_layer_activation[observation_index]
                                ,*third_layer_activation[observation_index]])
    return all_activations
```

In [21]:
```
observation_index = 0
all_activations = get_all_activations_index(observation_index)
show_activation(all_activations, 'Aktivierung für Index ' + str(observation_index))
```



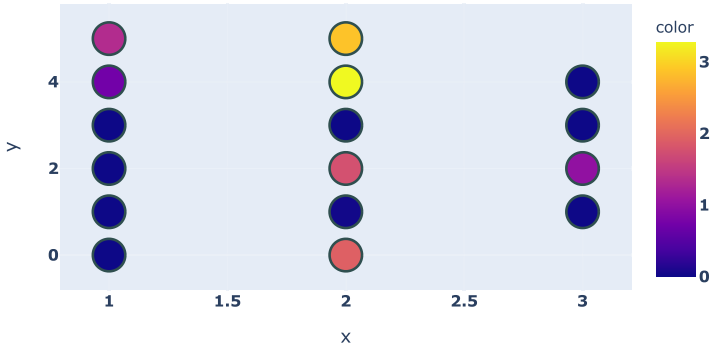
In [22]:
```
observation_index = 1
all_activations = get_all_activations_index(observation_index)
show_activation(all_activations, 'Aktivierung für Index ' + str(observation_index))
```

Aktivierung für Index 1



In [23]:
```
observation_index = 500
all_activations = get_all_activations_index(observation_index)
show_activation(all_activations, 'Aktivierung für Index ' + str(observation_index))
```

Aktivierung für Index 500

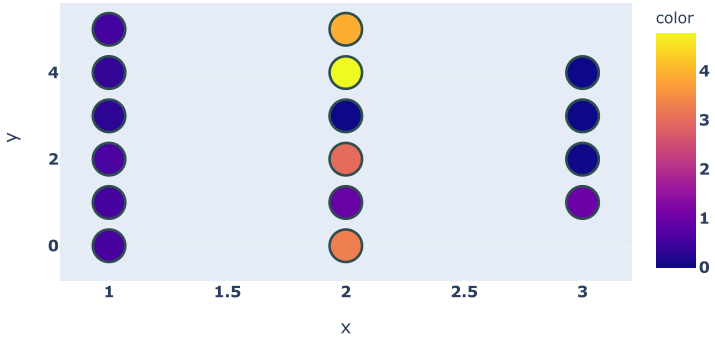


Darstellung für die mittlere Aktivierung der jeweiligen Klassen

In [24]:
```
def get_all_activations_filter(observation_filter):
    first = np.mean(first_layer_activation[observation_filter],axis=0)
    second = np.mean(second_layer_activation[observation_filter],axis=0)
    third = np.mean(third_layer_activation[observation_filter],axis=0)
    return *first, *second, *third
```

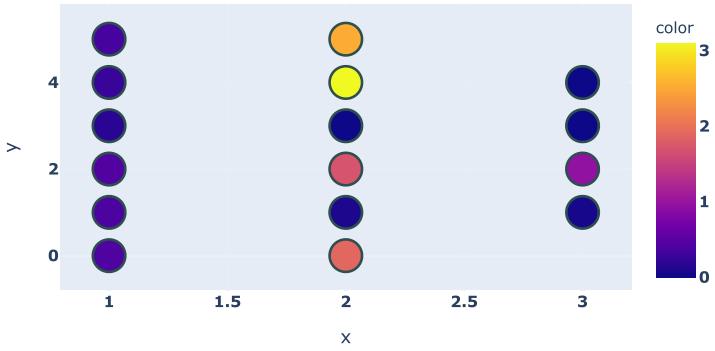
In [25]:
```
observation_filter =  df.index[df.label==0]

all_activations = get_all_activations_filter(observation_filter)

show_activation(all_activations, 'Aktivierung für Klasse = 0')
```

Aktivierung für Klasse = 0



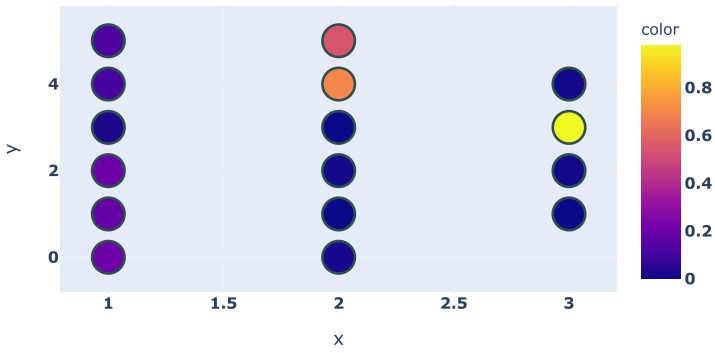
In [26]:
```
observation_filter =  df.index[df.label==1]

all_activations = get_all_activations_filter(observation_filter)

show_activation(all_activations, 'Aktivierung für Klasse = 1')
```

Aktivierung für Klasse = 1



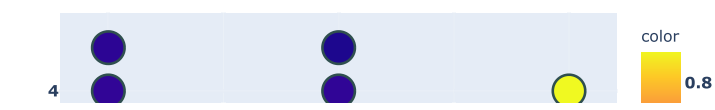
In [27]:
```
observation_filter =  df.index[df.label==2]

all_activations = get_all_activations_filter(observation_filter)

show_activation(all_activations, 'Aktivierung für Klasse = 2')
```

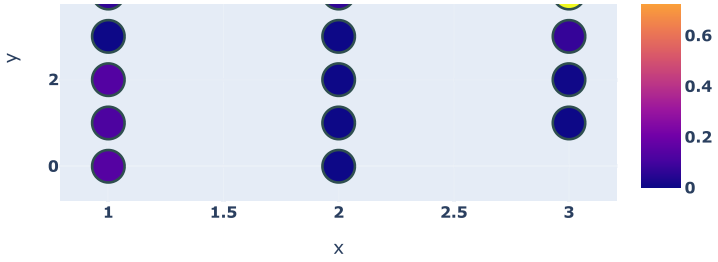
Aktivierung für Klasse = 2



In [28]:
```
observation_filter =  df.index[df.label==3]

all_activations = get_all_activations_filter(observation_filter)

show_activation(all_activations, 'Aktivierung für Klasse = 3')
```

Aktivierung für Klasse = 3

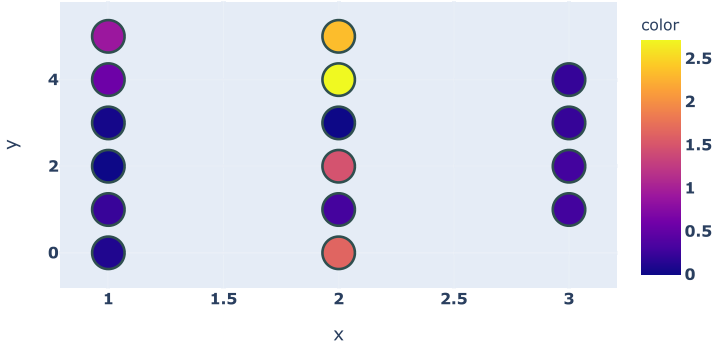




Darstellung für die mittlere Aktivierung der einzelnen Quadranten

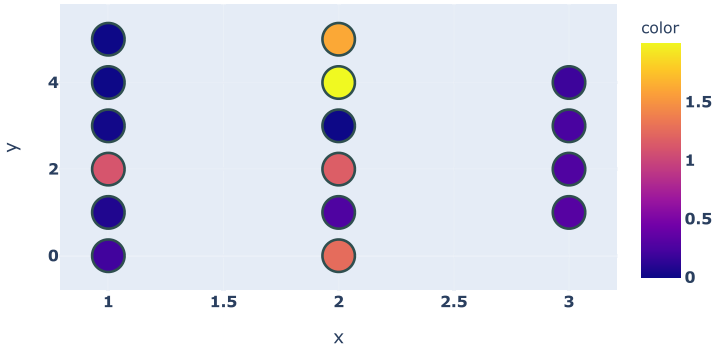
In [29]:
```
observation_filter =  (df.iloc[df.index[df.X0>=0]][df.X1>=0]).index

all_activations = get_all_activations_filter(observation_filter)

show_activation(all_activations, 'Aktivierung für alle Beispiele im oberen rechten Quadranten')
```

Aktivierung für alle Beispiele im oberen rechten Quadranten



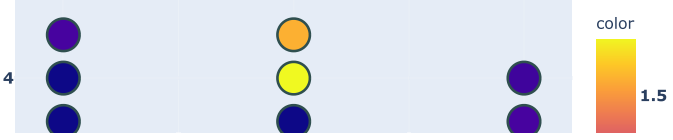
In [30]:
```
observation_filter =  (df.iloc[df.index[df.X0<=0]][df.X1<=0]).index

all_activations = get_all_activations_filter(observation_filter)

show_activation(all_activations, 'Aktivierung für alle Beispiele im unteren linken Quadranten')
```

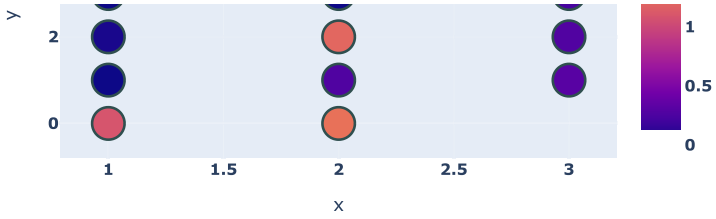
Aktivierung für alle Beispiele im unteren linken Quadranten



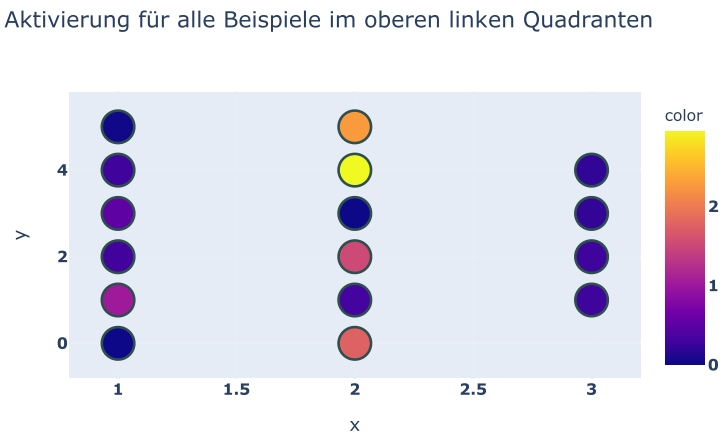
In [31]:
```
observation_filter =  (df.iloc[df.index[df.X0>=0]][df.X1<=0]).index

all_activations = get_all_activations_filter(observation_filter)

show_activation(all_activations, 'Aktivierung für alle Beispiele im unteren rechten Quadranten')
```

Aktivierung für alle Beispiele im unteren rechten Quadranten





In [32]:
```
observation_filter =  (df.iloc[df.index[df.X0<=0]][df.X1>=0]).index

all_activations = get_all_activations_filter(observation_filter)

show_activation(all_activations, 'Aktivierung für alle Beispiele im oberen linken Quadranten')
```



Zusammenfassend kann hier beobachtet werden, dass man die Unterschiede in der Reaktion des Netzes bezogen auf den Input sehen kann, weil das Netz eine überschaubare Größe hat. So kann beispielsweise abgeleitet werden, dass die zweite Schicht vermutlich etwas zu breit ist, da der Knoten #5 stets die gleiche (niedrige) Aktivierung aufzuweisen scheint, egal auf welche Weise die Daten unterteilt werden. Dies würde darauf hindeuten, dass keine erklärende Kraft damit verbunden ist.

Aufgabe 4: Partial Dependence (15 Punkte, Lernziele 5.3.5, 5.3.6)

Ein Partial Dependence Plot soll erzeugt werden. Wählen Sie geeignete Darstellungsarten für die Merkmale `X1` und `X2` und erzeugen Sie auch 2D-Plots sowie 3D-Plots für beide Merkmale gemeinsam. Diskutieren Sie, ob die 2D- oder 3D-Darstellung für zwei Merkmale informativer ist.

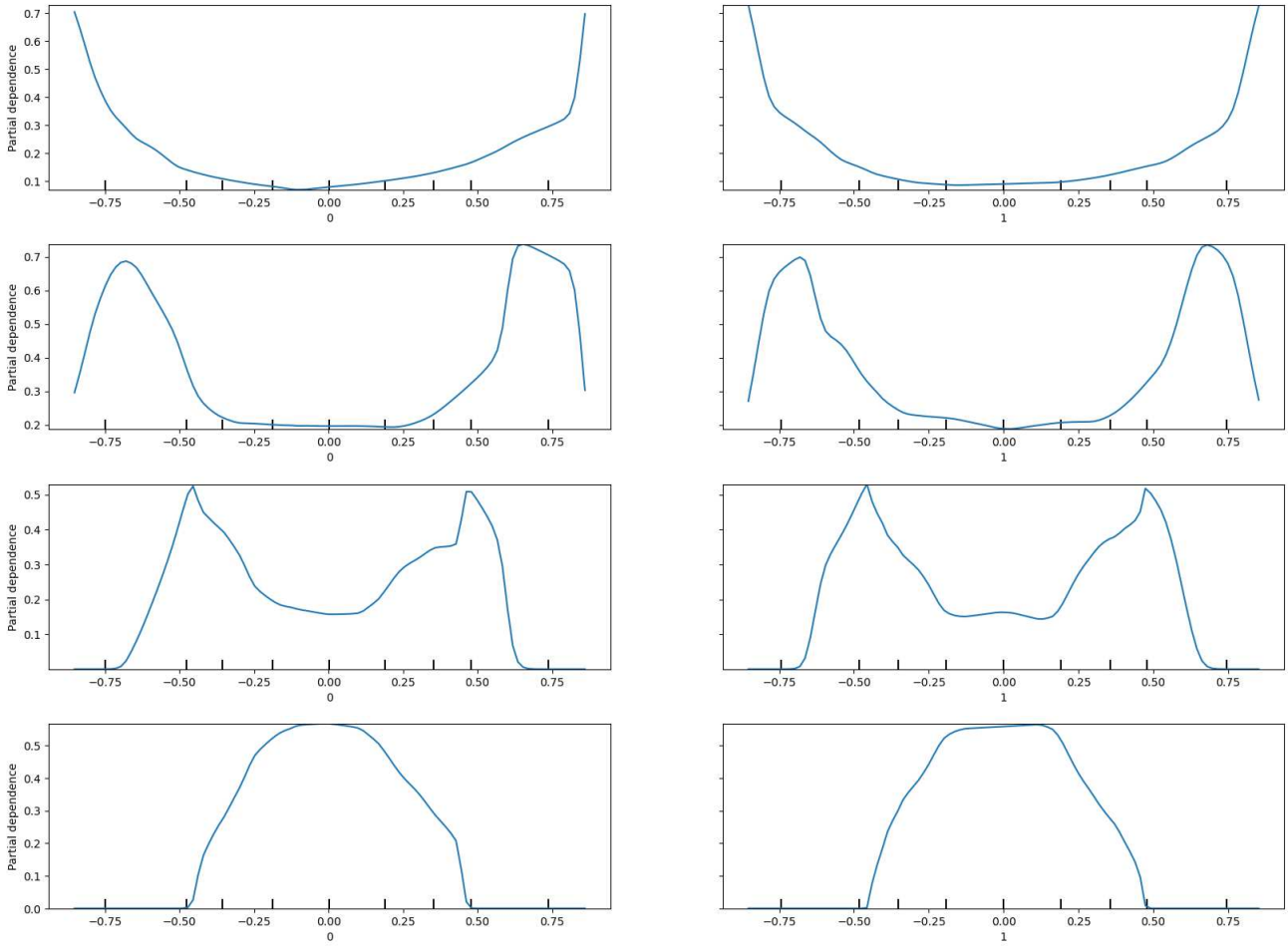
Sind PDPs geeignet, die Wirkungsweise des erzeugten Classifiers zu illustrieren? Welche Schwierigkeiten zeigen sich dabei? Inwiefern kann ein PDP dabei helfen zu entscheiden, ob ein Modell einfacher oder komplexer werden muss?

In [33]:
```
features = [0, 1] # Angabe, für welche Merkmale PDPs erzeugt werden sollen.
plt.rcParams["figure.figsize"] = (20,3) # Größe der Plots auf eine brauchbare Dimension setzen.
```

Nach dem Import der nötigen Komponenten wird das von oben bekannte Netz noch einmal mit einem Wrapper trainiert, damit man für den Classifier die scikit-learn-API verwenden kann. Dies ist notwendig, um ohne Modifikationen bspw. Partial-Dependence-Plots mit scikit-learn zu erzeugen.

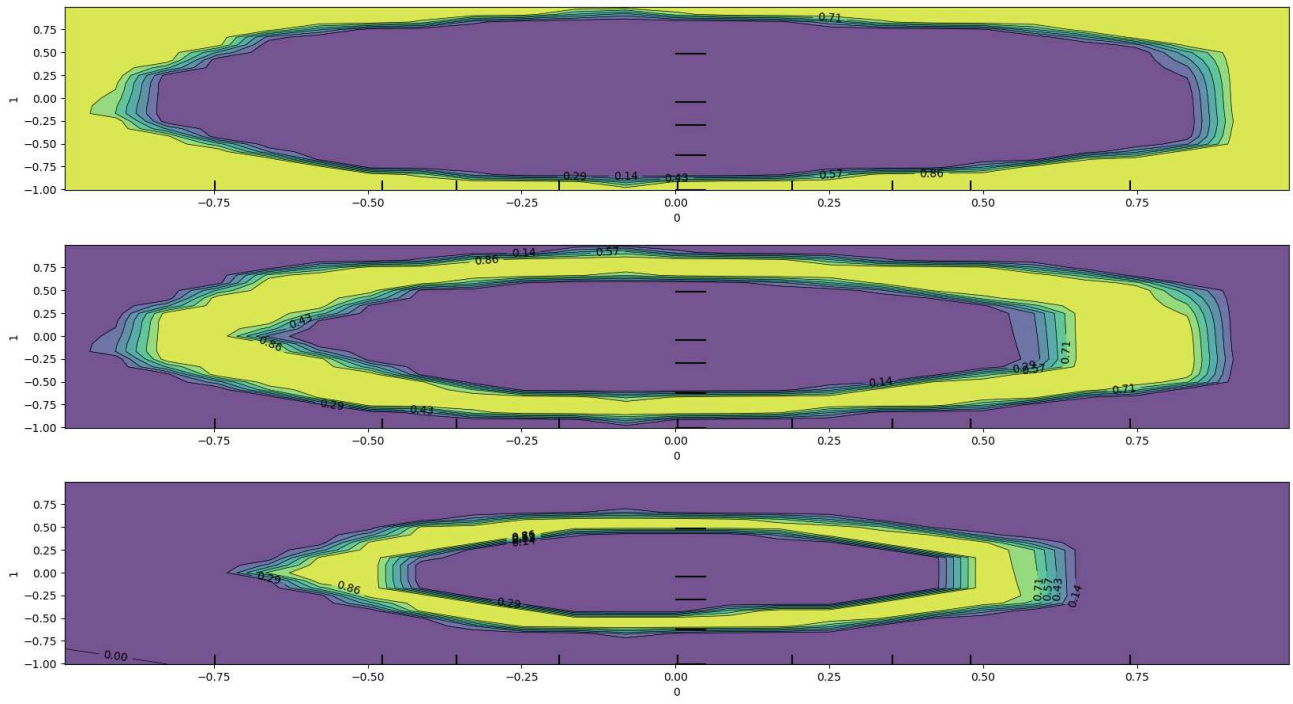
In [34]:
```
def circ_class_wrap():
    model = keras.Sequential([
        keras.layers.Dense(6, activation='relu', input_shape=[len(X.shape)]),
        keras.layers.Dense(6, activation='relu'),
        keras.layers.Dense(4, activation='softmax')
        ])

    model.compile( loss= tf.keras.losses.categorical_crossentropy,
                optimizer = tf.keras.optimizers.Adam(lr = 0.01),
              metrics=['accuracy'])

    return model
```

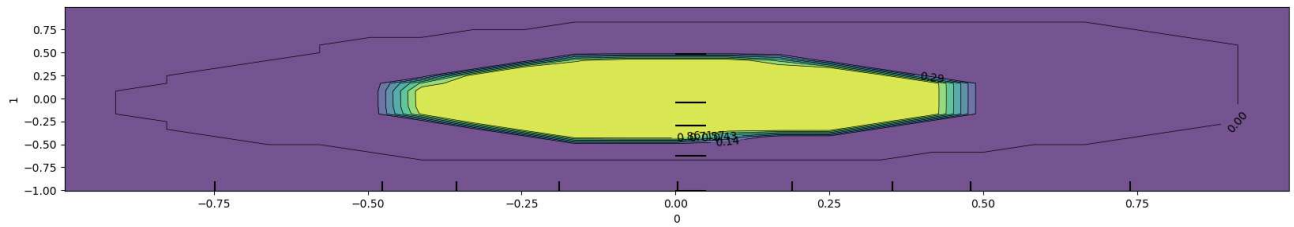
In [35]:
```
callable_classifier = KerasClassifier(build_fn=lambda:circ_class_wrap(),verbose=0)
callable_classifier._estimator_type = "classifier"
callable_classifier.fit(X,y, epochs=100, validation_split = 0.2)
callable_classifier.dummy_ = None
callable_classifier.name_ = 'classifier'
pdp_plot = PartialDependenceDisplay.from_estimator(callable_classifier,X,features,target=0)
pdp_plot = PartialDependenceDisplay.from_estimator(callable_classifier,X,features,target=1)
pdp_plot = PartialDependenceDisplay.from_estimator(callable_classifier,X,features,target=2)
pdp_plot = PartialDependenceDisplay.from_estimator(callable_classifier,X,features,target=3)
plt.show()
```



Aus den 1D-Darstellungen ist gut zu erkennen, dass im Gegensatz zu häufig eher monotonen Abhängigkeiten wie bei aktuariellen Risikomerkmalen die Decision Boundaries hier stark nichtlinear ausfallen und die bimodale Verteilung der Klassen in Bezug auf die jeweilige Achse noch einmal unterstreichen. Darüber hinaus sieht man die leichten Anomalien in der Verteilung durch das Rauschen der spezifischen Positionen der Beispiele. Anhand der "Glattheit" der PDPs lässt sich ablesen, dass zusätzliche Freiheitsgrade möglicherweise eine etwas bessere Anpassungsgüte erreichen könnten. Grundsätzlich ist aber nicht davon auszugehen, dass das erzeugte Modell entscheidende Probleme aufweist, denn die PDPs sind bereits relativ glatt.

In [36]:
```
features = [(0,1)]
for class_outcome in range(4):
    pdp_plot = PartialDependenceDisplay.from_estimator(callable_classifier
                                                       ,X
                                                       ,features,target=class_outcome
                                                       ,grid_resolution=25
                                                       ,percentiles=(0.01,0.99)
                                                       ,n_jobs=1)
    plt.show()
```

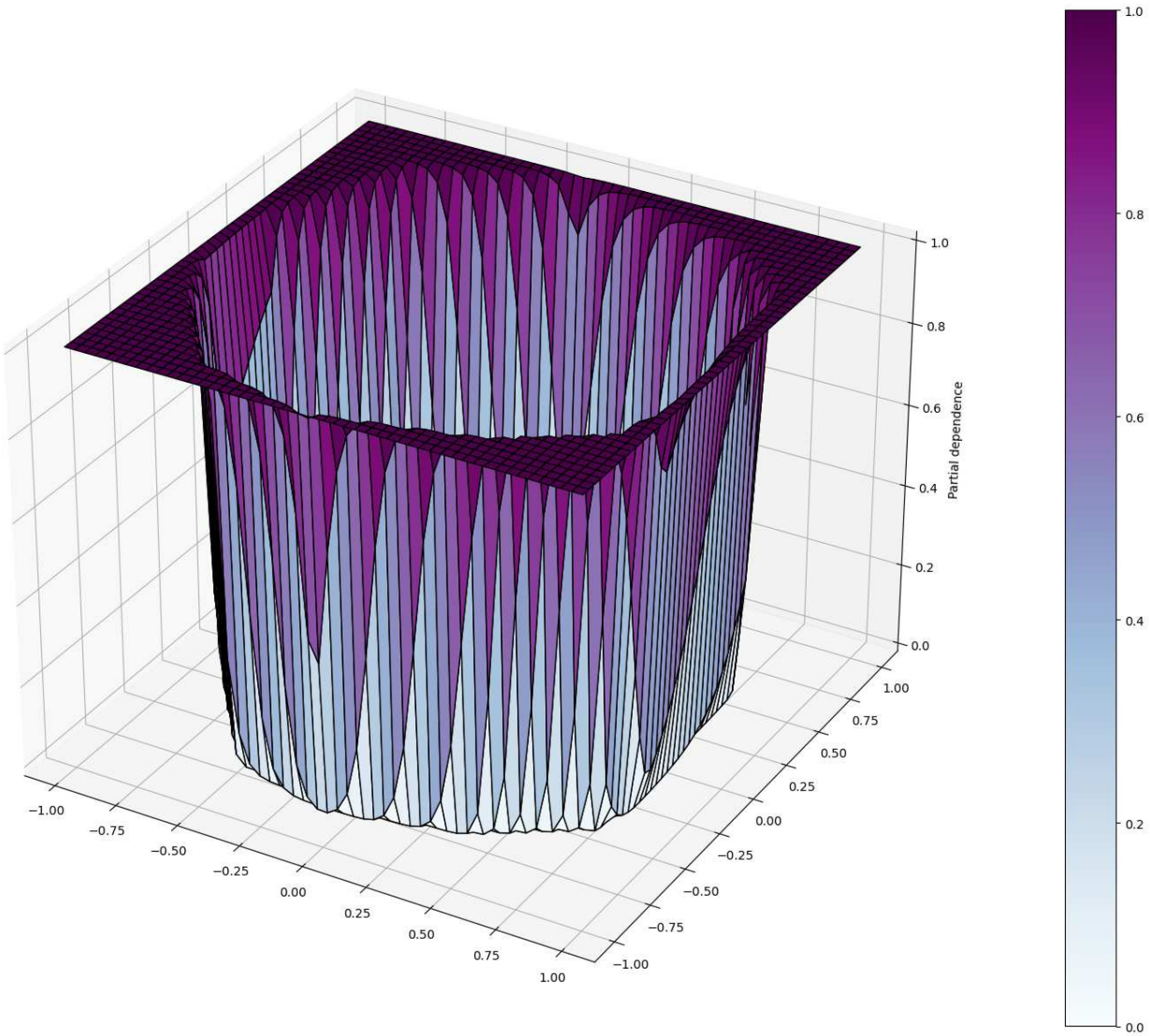




In [37]:
```python
def show_PDP_classifier(Klasse):
    gr = 50 # Grid Resolution
    fig = plt.figure(figsize=(30,15))
    features = [0,1]
    pdp = partial_dependence(callable_classifier
                             , (df[df['label'] == Klasse])[['X0','X1']]
                             , features=features
                             , kind="average"
                             , grid_resolution=gr)

    XX, YY = np.meshgrid(pdp["values"][0], pdp["values"][1])
    Z = pdp.average[0].T

    ax = fig.add_subplot(projection="3d")
    fig.add_axes(ax)

    surf = ax.plot_surface(XX, YY, Z, rstride=1, cstride=1, cmap=plt.cm.BuPu, edgecolor="k")
    ax.set_zlabel("Partial dependence")
    plt.colorbar(surf)
    plt.suptitle("Partial Dependence Plot für Klassifizierung: Klasse " + str(Klasse))
    plt.show()
```

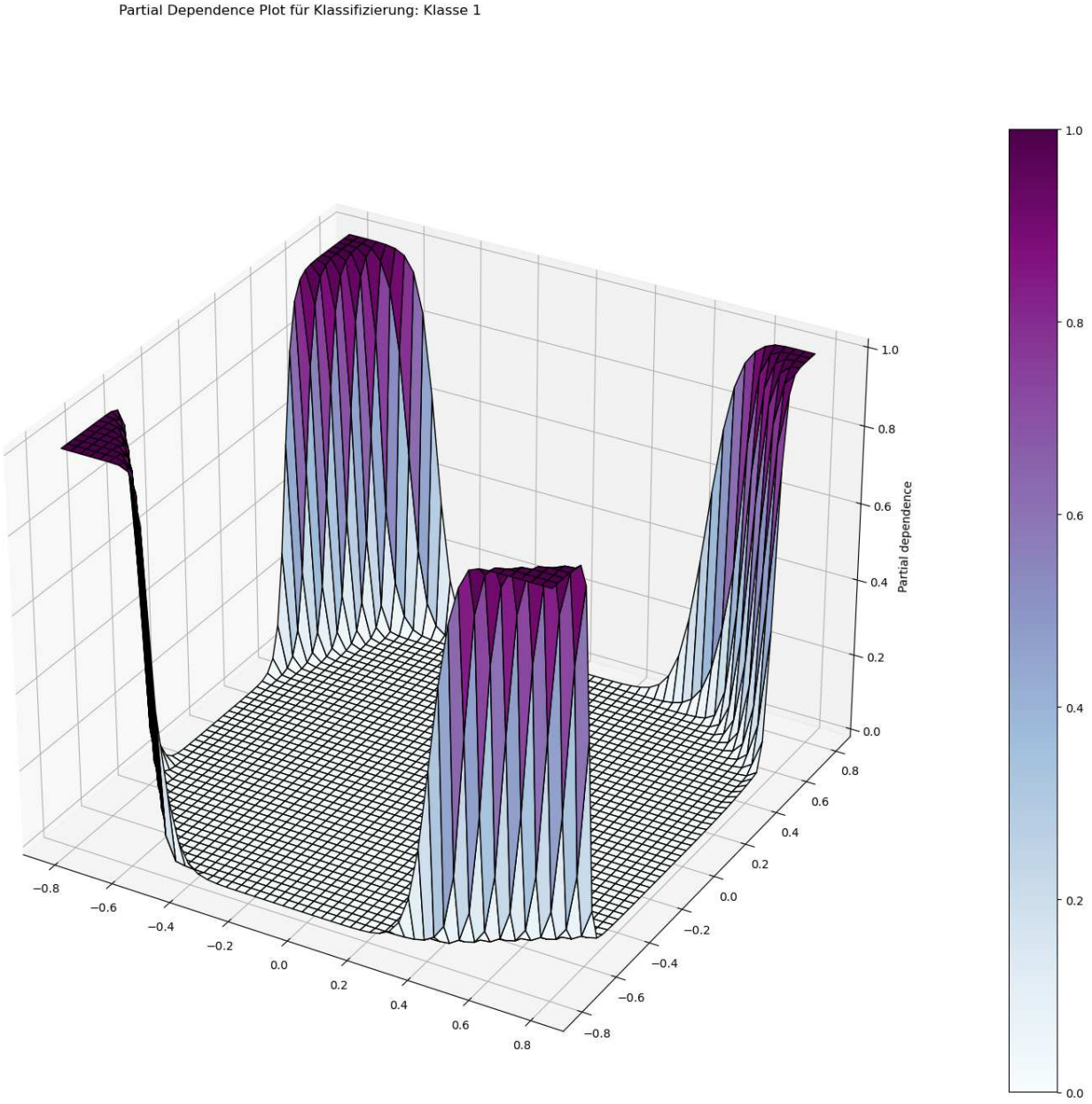
In [38]:
```python
Klasse = 0
show_PDP_classifier(Klasse)
```

Partial Dependence Plot für Klassifizierung: Klasse 0

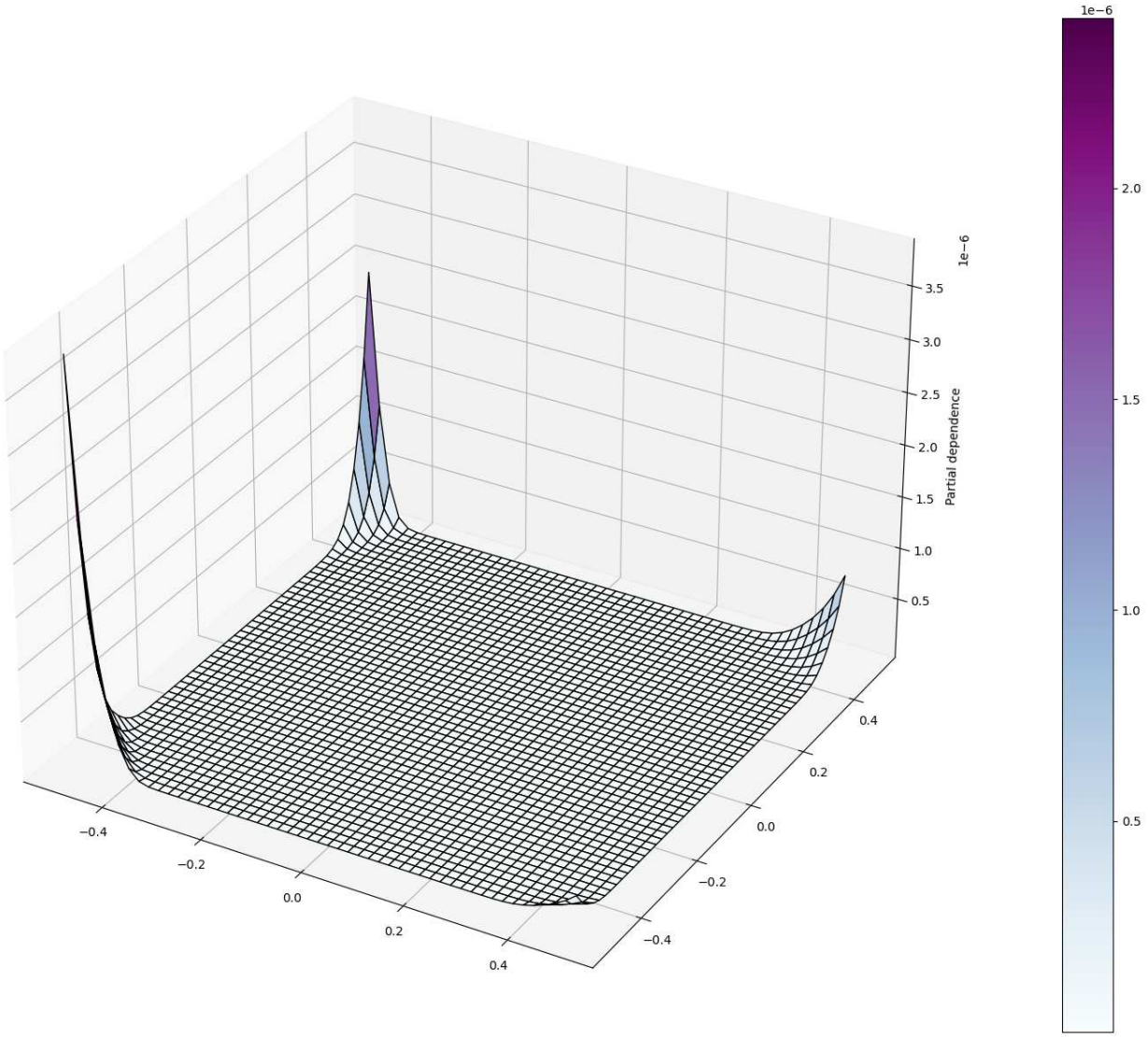


In [39]:
```python
Klasse = 1
```

```
show_PDP_classifier(Klasse)
```

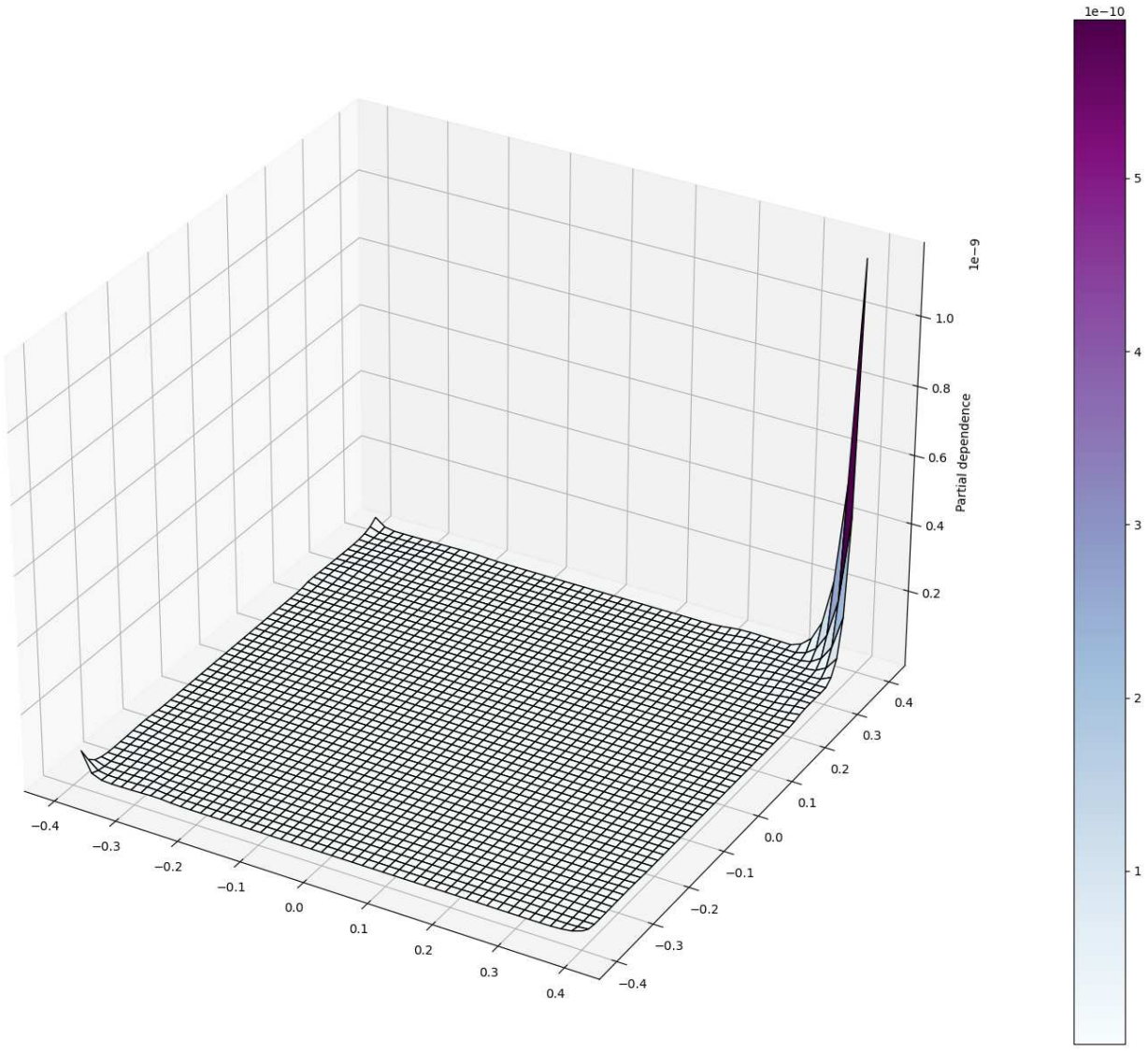


In [40]:
```
Klasse = 2
show_PDP_classifier(Klasse)
```



In [41]:
```
Klasse = 3
show_PDP_classifier(Klasse)
```

Partial Dependence Plot für Klassifizierung: Klasse 3



Eine wesentliche Eigenschaft des Classifiers besteht darin, dass die jeweiligen Klassengrenzen zwischen den Kreisen sehr scharf sind, bzw. sein müssen. Im Verhältnis zum "Merkmalsquadrat" [[-1,1],[-1,1]] sind daher nur jeweils kleine Bereiche der Darstellung wirklich relevant. Wenngleich dies ein Stück weit Geschmackssache ist, wirkt die 2-dimensionale Darstellung tendenziell ruhiger und bietet bei geeigneter Farbskala einen besseren Kontrast. Hinzu kommt, dass bei einer 3-dimensionalen Darstellung das Risiko besteht, dass die Betrachtungsperspektive ablenkend wirkt oder relevante Details verdeckt.

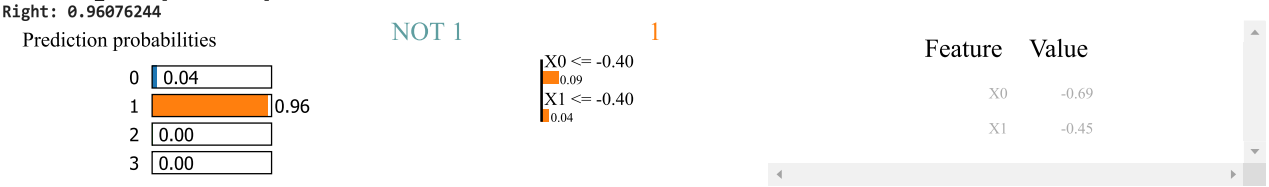
5. LIME (5 Punkte, Lernziele 5.3.5, 5.3.6)

Erstellen Sie LIME-Erklärungen für die Beobachtungen 0, 1 und 500 und erläutern Sie, inwiefern der LIME-Ansatz hier zum Verständnis des Modells beiträgt.

In [42]:
```
feature_names=['X0','X1']
explainer = lime.lime_tabular.LimeTabularExplainer(X, mode='classification',training_labels=df['label'],feature_names=feature_names,verbos
```

In [43]:
```
exp = explainer.explain_instance(X[0],model.predict, num_features=3)
exp.show_in_notebook(show_all=True)
```

```
Intercept 0.3169969806192023
Prediction_local [0.44805199]
Right: 0.96076244
```

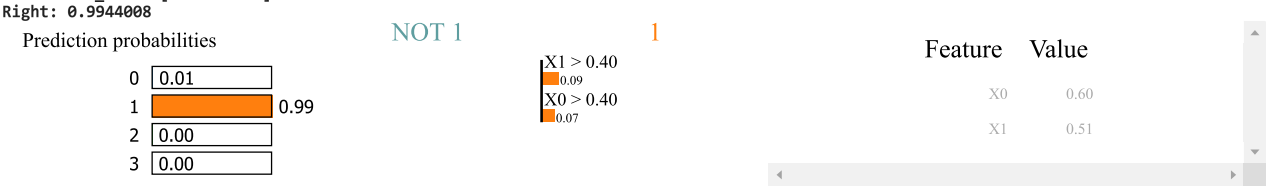


In [44]:
```
exp = explainer.explain_instance(X[1],model.predict, num_features=3)
exp.show_in_notebook(show_all=True)
```

```
Intercept 0.40612356106591585
Prediction_local [0.15946509]
Right: 0.0035562401
```

In [45]:
```
exp = explainer.explain_instance(X[500],model.predict, num_features=3)
exp.show_in_notebook(show_all=True)
```

```
Intercept 0.3100470376039104
Prediction_local [0.47034469]
Right: 0.9944008
```



Dass das *L* in LIME für 'linear' steht, zeigt sich an diesem Beispiel in ganz besonders deutlicher Weise: Die Annahme, dass es eine lineare Abhängigkeit von den Merkmalen gäbe, ist durch die Ringstruktur zwar lokal erfüllt, insgesamt aber komplett falsch. Dies führt auch dazu, dass die Einschätzung des Lime-Explainers stets mit dem Vorzeichen des jeweiligen Merkmals korreliert. Dies ist zwar grundsätzlich für jede einzelne Beobachtung richtig, hilft aber nicht beim Verständnis von anderen Beobachtungen. Streng genommen müsste also eine Inspektion, die nur auf LIME beruht, scheitern, wenn nicht wirklich alle Beobachtungen explizit betrachtet werden, bzw. eine sehr große Stichprobe vorliegt. Die pauschale Aussagen der Form "das Merkmal ist [positiv/negativ], also ist die Vorhersage der Klasse 0-3 [positiv/negativ] beeinflusst ist daher in jedem Falle irreführend.

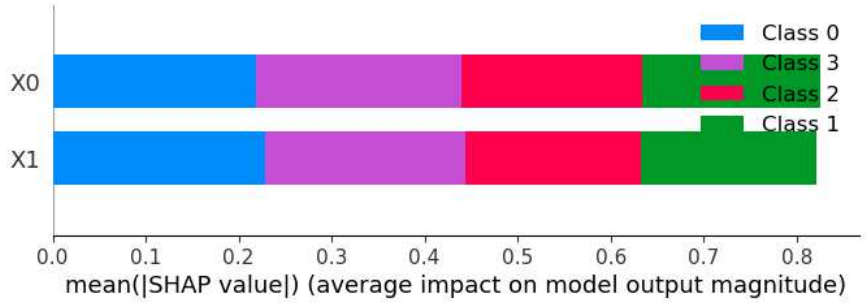
6. Shapley-Values (10 Punkte, Lernziele 5.3.5, 5.3.6)

Führen Sie eine Erklärbarkeitsuntersuchung mit Hilfe von Shapley-Values durch. Erzeugen Sie `summary` -, `dependence` - und `force` -Plots. Ordnen Sie die Eignung der Methode zur Erlagungen von Verständnis in Bezug auf das Basismodell ein, auch in Abgrenzung zu den vorigen Aufgaben (PDP, LIME).

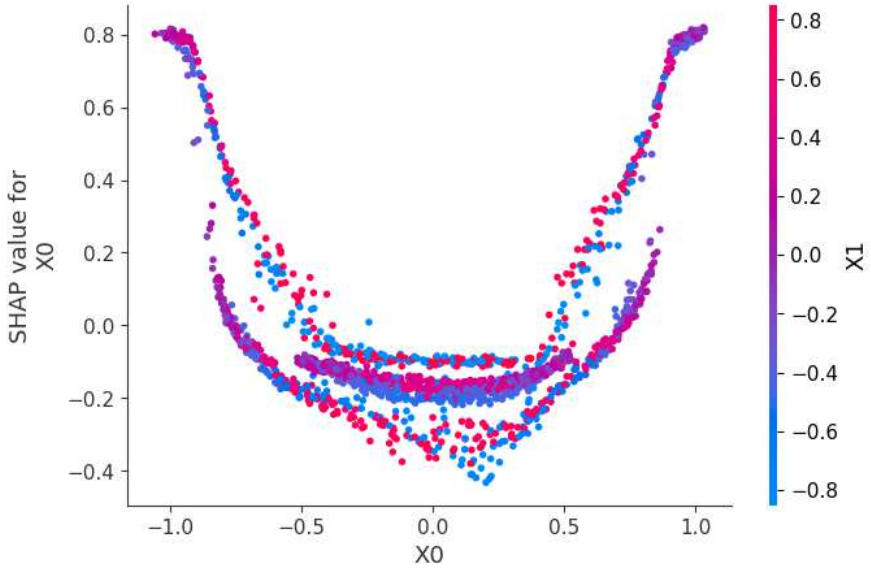
In [46]:
```
shap.initjs()
```

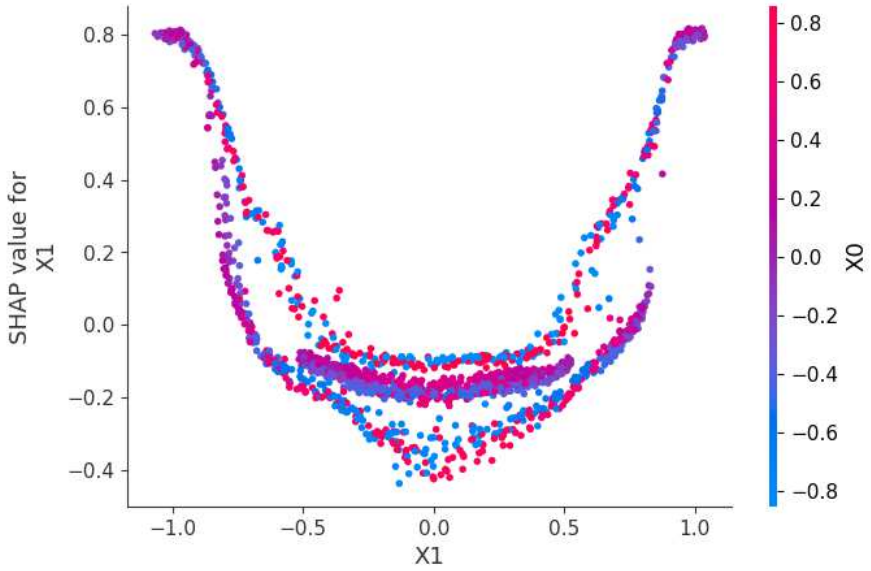
In [47]:
```
shap_values = shap.explainers.Sampling(model.predict
                                       ,data = pd.DataFrame(X,columns=['X0','X1'])
                                       ,check_additivity = True).shap_values(X)
```

In [48]:
```
shap.summary_plot(shap_values, plot_type="bar",feature_names=feature_names)
```



In [49]:
```
shap.dependence_plot(0, shap_values[0], X, feature_names=feature_names)
shap.dependence_plot(1, shap_values[0], X, feature_names=feature_names)
```



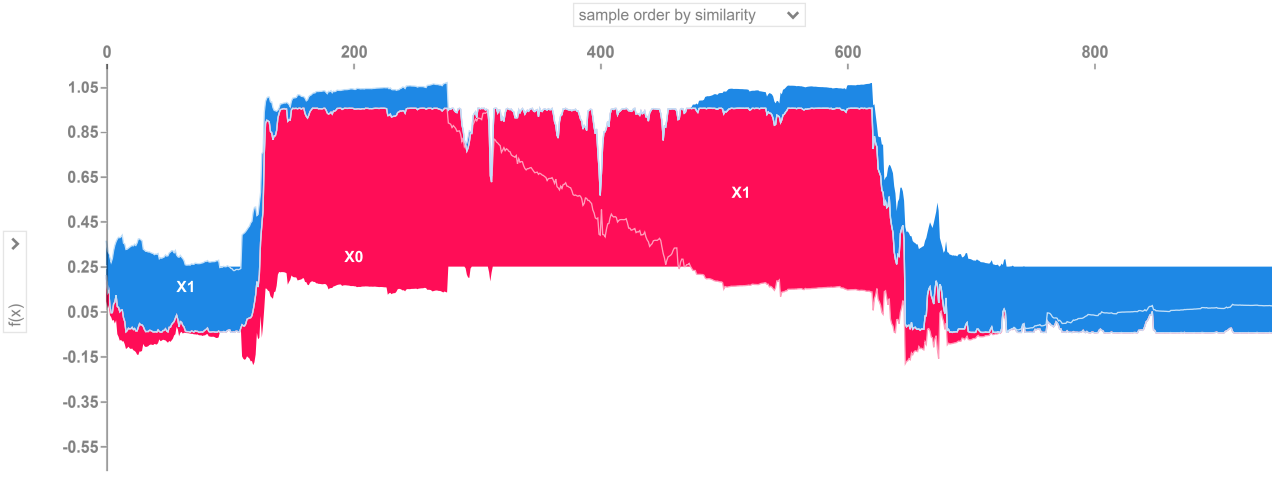


Die sog. Dependence-Plots erlauben uns hier die Klassenverteilung in Bezug auf die beiden Merkmale zu untersuchen. Hier erhalten wir in gewisser Weise das Komplement zum Scatterplot von Aufgabenteil 1: Im Gegensatz zur Darstellung basiert auf den Werten sind die Beispiele hier nach ihrem Impact im Modell aufgetragen. Man kann so gut erkennen, dass an den fernen Rändern des Zustandsraumes die Entscheidungsgrundlage stärker ist, also die äußeren Ringe besser getrennt werden. Dies ist abgesehen vom hier konstruierten Datensatz eine eher untypische Beobachtung, da normalerweise Beobachtungen am Rand des Wertebereichs selten und somit mit starker Unsicherheit behaftet sind.

In [50]:
```
# Erzeuge globale Erwartungswerte
expected = np.mean(model.predict(pd.DataFrame(X,columns=feature_names)))
```

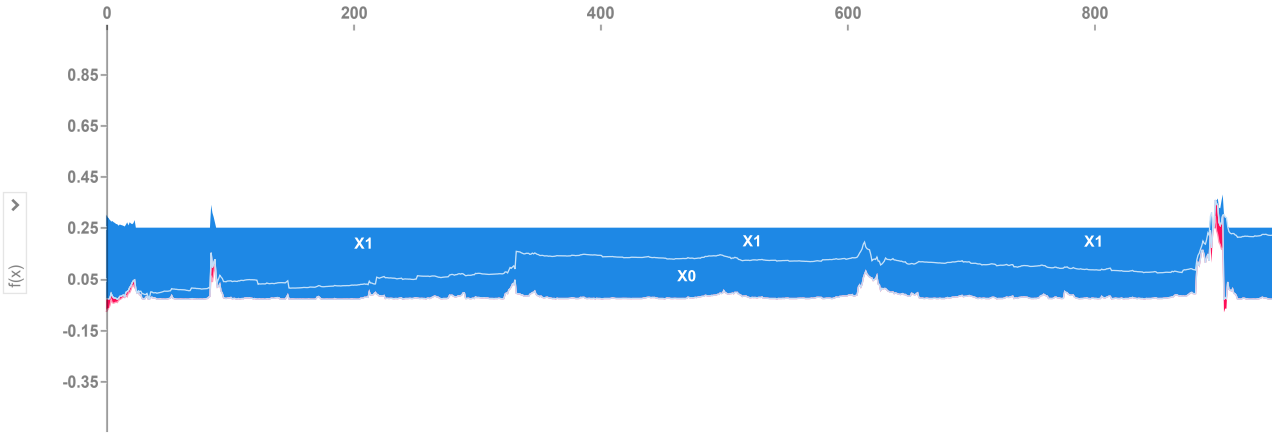
In [51]:
```
# Plotte Force-Plots
shap.force_plot(expected, shap_values[0], X, feature_names=feature_names)
```

Out[51]:



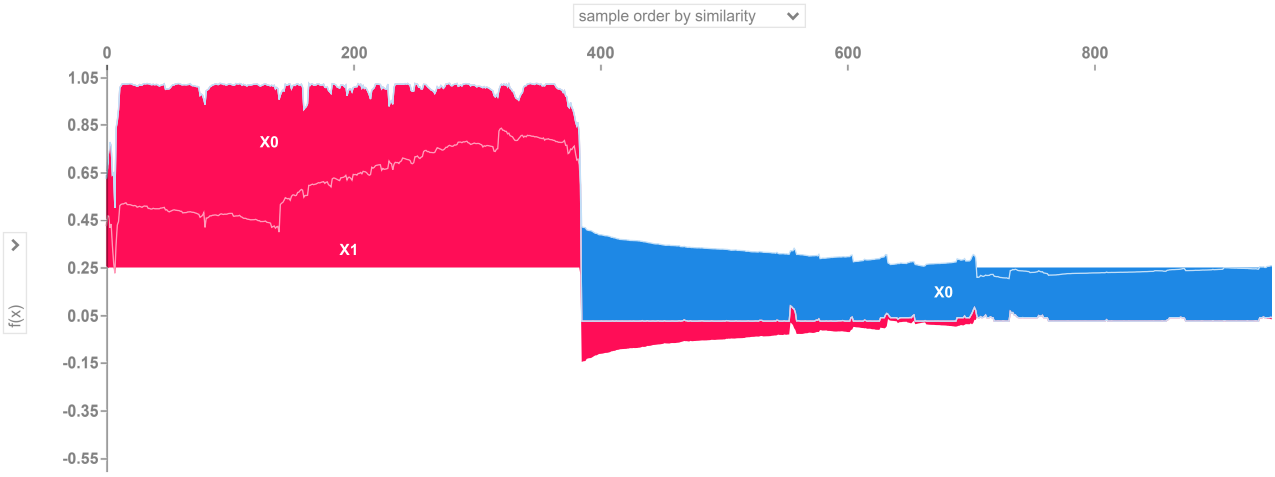
In [52]:
```
# Plotte Force-Plots
shap.force_plot(expected, shap_values[1], X, feature_names=feature_names)
```

Out[52]:



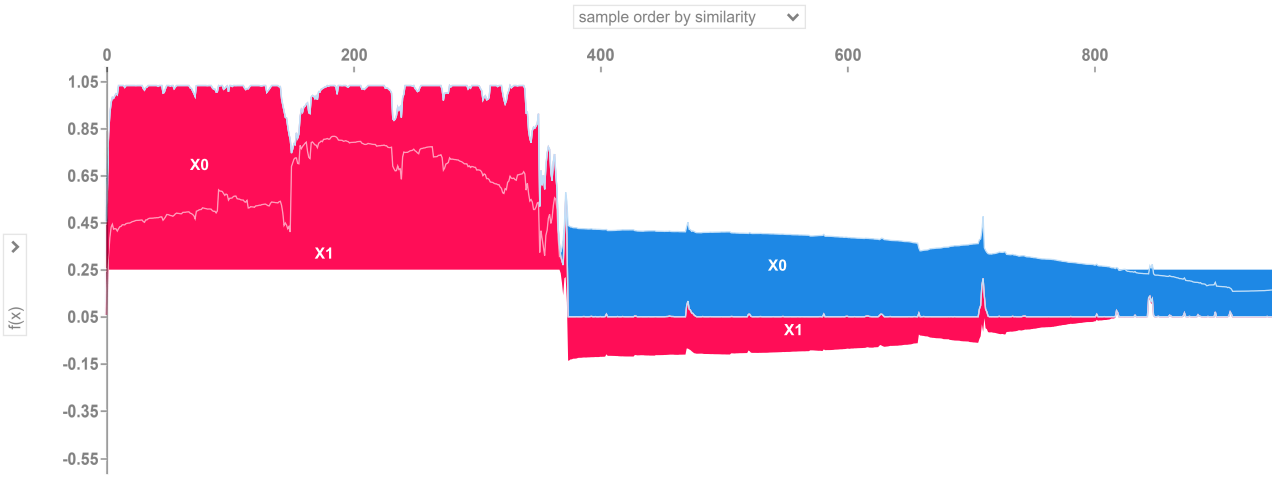
In [53]:
```
# Plotte Force-Plots
shap.force_plot(expected, shap_values[2], X, feature_names=feature_names)
```

Out[53]:



In [54]:
```
# Plotte Force-Plots
shap.force_plot(expected, shap_values[3], X, feature_names=feature_names)
```

Out[54]:



Mit Hilfe der Force-Plots können gleich mehrere Aspekte untersucht werden. Zum einen replizieren sie die Partial Dependence-Plots von Aufgabenteil 5, zum anderen lässt sich anhand `order by similarity` die Verteilung der Vorhersagergebnisse betrachten. Dies erlaubt im Gegensatz zu LIME eine tiefergehende Betrachtung von nicht-linearen Effekten, löst mindestens dabei auf, dass kein monotoner, linearer Zusammenhang besteht.

Lösungsvorschlag zur Prüfungsaufgabe Completion, Block B: Textklassifikation [105 Punkte]

Contents

Verwendete Quellen:

1. van Gils & Nagelkerke (2020), "Natural Language Processing for predictive purposes with R", https://medium.com/broadhorizon-cmotions/natural-language-processing-for-predictive-purposes-with-r-cb65f009c12b

2. Merz & Wüthrich (2022), "Statistical Foundations of Actuarial Learning and its Applications", "https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3822407"

3. D. Falbel (2017), "Word Embeddings with Keras", https://blogs.rstudio.com/ai/posts/2017-12-22-word-embeddings-with-keras/

4. S. Glander (2019), "How to prepare data for NLP (text classification) with Keras and TensorFlow", https://www.r-bloggers.com/2019/01/how-to-prepare-data-for-nlp-text-classification-with-keras-and-tensorflow/

5. F. Chollet (2018), "Deep Learning with"Python" bzw. Chollet & Allaire (2018), "Deep Learning with R" (Buch)

6. Weitere: https://dk81.github.io/dkmathstats_site/rtext-freq-words.html,

Aufgabenspezifische bzw. technische Vorbemerkungen:

- Alternativ kann in Aufgabe B7 anstelle der LSTM-Layer GRU-Layer, dann aber durchgängig, verwendet werden.
- Bei Verwendung mehrerer, insbesondere ähnlicher Programmbibliotheken können "name-space"-Konflikte auftreten und müssen geeignet beseitigt werden
- Bei Verwendung insbesondere älterer Code-Vorlagen aus öffentlichen Quellen können aufgrund von Versionsänderungen Anpassungen erforderlich sein (z.B. Unterschiede TensorFlow 1 vs. 2)
- Zur Orientierung: Die gesamte Laufzeit des zur Aufgabe entwickelten Lösungsvorschlages beträgt auf einem betagten Notebook weniger als 10 Minuten. In diesem Laufzeitbudget ist eine Prognosegenauigkeit von (ggf. knapp) 90% erreichbar. Das PDF-Dokument ist einschließlich Output kleiner als 1 MB.

Block B: Notebook Textklassifikation

Die Datenbasis Schadenbeschreibung_Sparte-de.csv enthält zwei durch Semikolon getrennte Merkmale: 1. 'Description': kurze englischsprachige Schadenbeschreibung 2. 'line': Zuordnung zu einer Sparte/Risikoart bzw. Sonstige (Misc). Als Business Case soll auf Basis der Schadenbeschreibungen mittels Textmining eine automatische Spartenzuordnung entwickelt werden.

0: Benötigte Bibliotheken einbinden

```
suppressPackageStartupMessages({
  library(textstem)
  library(tm)
  library(tidytext)
  library(tidyverse)
  library(caret)
  library(uwot)
  library(tensorflow)
  library(keras)
  })

# Reproduzierbarkeit der Ergebnisse (u.a. TensorFlow 2.x, Python)
seed <- 42
set.seed(seed)
Sys.setenv(PYTHONHASHSEED = seed)
reticulate::py_set_seed(seed)
tensorflow::tf$random$set_seed(seed)
```

```
## Loaded Tensorflow version 2.8.0
```

1. Daten einlesen und visualisieren

Aufgabe B1: [Lernziel 6.1] [5 Punkte] Lesen Sie die Datei Schadenbeschreibung_Sparte-de.csv ein und vergewissern Sie sich, dass alle Datensätze korrekt verarbeitet wurden. Erstellen Sie eine Häufigkeitstabelle der Sparten (line) und zeigen Sie diese graphisch an. Geben Sie

zusätzlich die drei längsten Beschreibungen ("Description", gemessen an der Anzahl Zeichen) aus.

Lösungsvorschlag:

```
dat <- read.csv2("Schadenbeschreibung_Sparte-de.csv",stringsAsFactors = FALSE)
dim(dat)
```

```
## [1] 7194    2
```

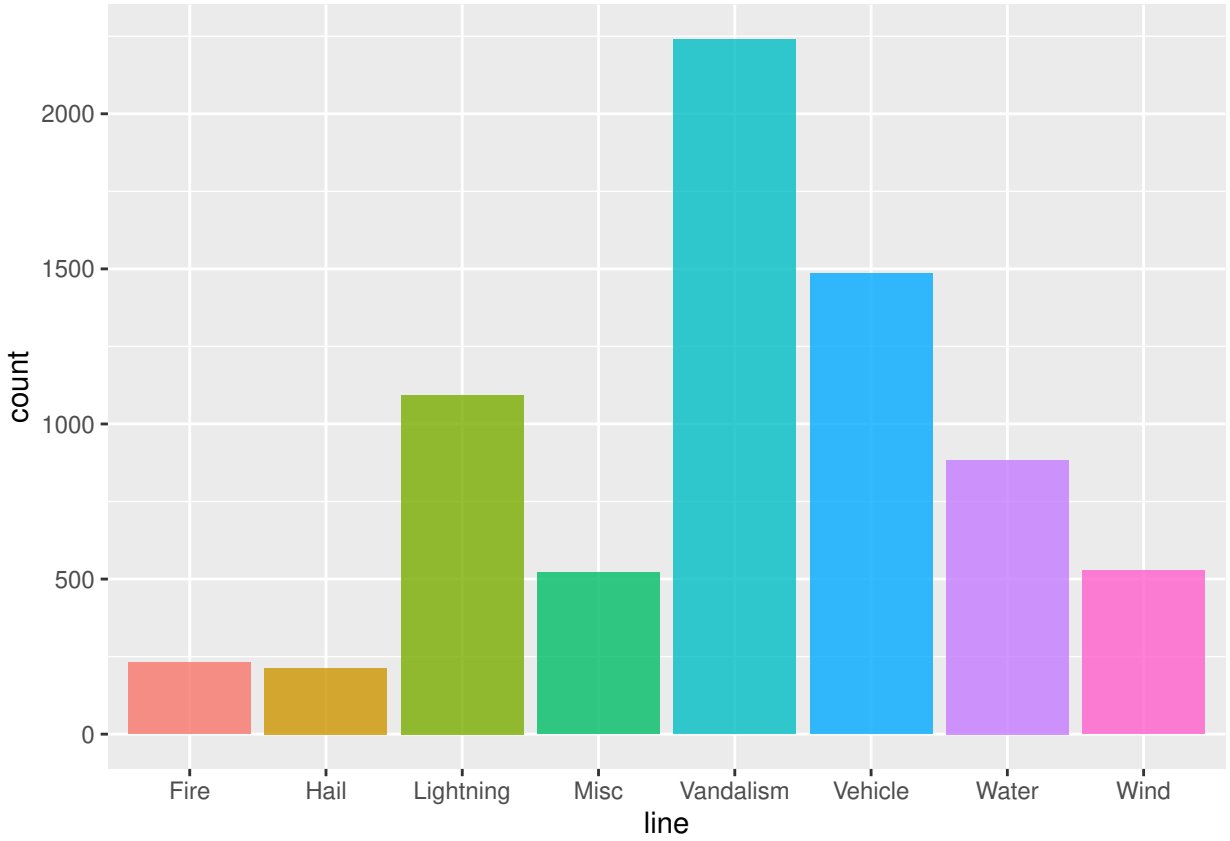
```
head(dat)
```

```
##                                        Description      line
## 1   windows broken at Lakeview Tech School           Vandalism
## 2 street light damaged                                   Wind
## 3 vehicle struck streetlamp; bike rack; tree           Vehicle
## 4 wind damage to roof                                    Wind
## 5 electrical fire                                        Fire
## 6 wind damage-Museum                                     Wind
```

```
# Sparten: Anzahl der Schäden je Sparte
table(dat$line)
```

```
##
##      Fire      Hail Lightning      Misc Vandalism   Vehicle     Water      Wind
##       231       213      1093       521      2240      1484       884       528
```

```
# Sparten: Graphische Ausgabe
dat %>%
  ggplot(aes(x = line, fill = line)) +
  geom_bar(alpha = 0.8) +
  guides(fill = "none")
```



Ausgabe der längsten Beschreibungen, gemessen an der Anzahl Zeichen (ohne führende oder nachfolgende Leerzeichen):

```
dat$lang <- nchar(trimws(dat$Description))
tail(dat[order(dat$lang),],3)
```

```
##                                                          Description
## 6400    Lake View elementary school spray paint windows-tree trunks
## 1962 water damage to rooms L12; L14 at Columbus elementary school
## 6129 Olson  elementary school - llm008119 - vehicle damaged bench
##           line lang
## 6400 Vandalism   59
## 1962     Water   60
## 6129   Vehicle   60
```

2. Schadenbeschreibungen aufbereiten

Aufgabe B2: [Lernziel 5.1] [15 Punkte] Erläutern Sie kurz die Textmining-Schritte "Stoppwörter entfernen" und Lemmatisierung und nenne Sie zu beiden Begriffen ein kurzes Beispiel. Entfernen Sie aus dem Feld "Description" typische englischsprachige Stopwords und führen Sie eine Lemmatisierung durch. Zeigen Sie für eine kleine Auswahl an Datensätzen die ursprünglichen und die bereinigten Texte an. Ermitteln Sie die Häufigkeit der verbleibenden Wörter und zeigen Sie die Verteilung für alle Wörter, die mindestens 250-mal vorkommen, in einem Balkendiagramm absteigend sortiert an. Das mit Abstand häufigste Wort ist "damage" (nach Lemmatisierung). Ermitteln Sie je Sparte (line) den Anteil der Schadenbeschreibungen,

bei denen das Wort "damage" mindestens einmal vorkommt. Interpretieren Sie das Ergebnis und wägen Sie ab, ob "damage" als Stopword entfernt werden soll.

Lösungsvorschlag:

Im Textmining-Schritt "Stoppwörter entfernen" werden bestimmte Wörter für die Weiterverarbeitung ausgeschlossen. Die Stoppwörter sind (in jeder Sprache) häufig vorkommende Wörter wie z.B. Artikel, Präpositionen etc. die für den eigentlichen Inhalt keine Bedeutung haben. In den vorliegenden englischen Beschreibungen sind das z.B. Wörter wie "the", "a", "an", "so" etc. Durch die Entfernung dieser Wörter kann sich das Modell auf die eigentlich wichtigen Informationen konzentrieren, zudem ergeben sich für das Training keine negativen Konsequenzen.

Die Lemmatisierung ist beim Text Mining notwenig, um zusammengehörende Wörter gruppieren zu können. Bei der Lemmatisierung orientiert man sich an Listen bzw. Datenbanken, die reflektierte Formen enthalten. So werden z.B die Wörter "gelaufen" und "laufen" beiden mit dem Wort "laufen" identifiziert.

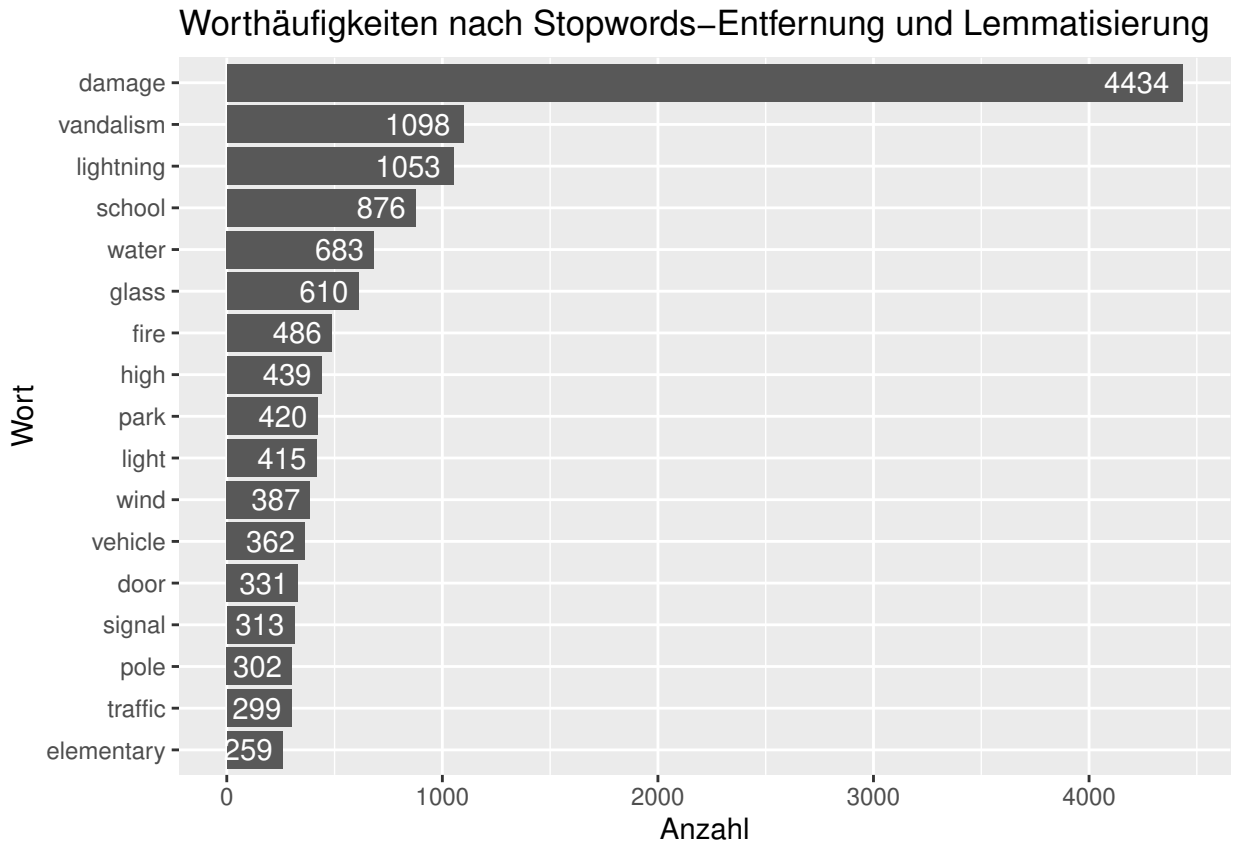
Entfernung von "Stopwords", Lemmatisierung und Vorher-Nachher-Vergleich einiger Schadenbeschreibungen:

```
text.clean <- removeWords( dat$Description , stopwords("english"))
text.clean <- lemmatize_strings(text.clean , dictionary = lexicon::hash_lemmas )
dat$clean <- text.clean
# Auswahl ursprüngliche vs bereinigte Texte (die nicht gleich sind)
head(dat[which(trimws(dat$Description) != trimws(dat$clean)),c(1,4)],7)
```

```
##                                         Description
## 1     windows broken at Lakeview Tech School
## 2   street light damaged
## 3   vehicle struck streetlamp; bike rack; tree
## 4   wind damage to roof
## 6   wind damage-Museum
## 7 glass vandalism at high school
## 8   sewer back up damage
##                                       clean
## 1          window break Lakeview Tech School
## 2                        street light damage
## 3 vehicle strike streetlamp; bike rack; tree
## 4                           wind damage roof
## 6                       wind damage - Museum
## 7               glass vandalism high school
## 8                          sewer back damage
```

Verbleibende Wörter zählen und anzeigen:

```
tibble(Text = dat$clean) %>%
  unnest_tokens(output = word, input = Text) %>%
  count(word, sort = TRUE) %>%
  filter(n >= 250) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  geom_col() +
  coord_flip() +
  labs(x = "Wort", y = "Anzahl",
       title = "Worthäufigkeiten nach Stopwords-Entfernung und Lemmatisierung") +
  geom_text(aes(label = n), hjust = 1.2, colour = "white")
```



Sonderbetrachtung “damage” (Anteil der Schadenbeschreibungen je Sparte, bei denen das Wort “damage” mindestens ein Mal vorkommt):

```
# Damage im Text suchen und Anteil je Klasse berechnen
dat$prop_damage <- with(dat,ifelse(grepl("damage",dat$Description),1,0))

# Textanteil mit "damage" je Sparte
aggregate(prop_damage ~ line, data=dat, FUN=function(x) {sum(x)/length(x)})
```

```
##        line prop_damage
## 1      Fire   0.7489177
## 2      Hail   0.9530516
## 3 Lightning   0.9304666
## 4      Misc   0.5009597
## 5 Vandalism   0.2080357
## 6   Vehicle   0.7553908
## 7     Water   0.7794118
## 8      Wind   0.7878788
```

Bei Vandalismus tritt das Wort “damage” stark unterdurchschnittlich häufig auf. Der Anteil liegt je nach Sparte zwischen 21% und 95%, unterscheidet sich also erheblich und könnte eine prädiktive Aussage haben. Daher wird hier darauf verzichtet, “damage” als (zu) häufiges Stoppwort zu entfernen.

3. Trainings- und Testdaten erzeugen, Tokenisierung vornehmen und überprüfen

Aufgabe B3: [Lernziel 5.1] [15 Punkte] Teilen Sie die Datensätze in 75% Trainingsdaten und 25% Testdaten über eine reproduzierbare Zufallsauswahl auf. Führen Sie an den bereinigten Schadenbeschreibungen die Schritte Tokenisierung, Sequenzbildung und "Padding" durch. Dabei soll die Betrachtung auf die häufigsten 100 Worte sowie Sequenzen der Länge 10 begrenzt werden. Die Datenstrukturen sind so aufzubereiten, dass mit Keras eine multinominale Klassifikation (der Sparten) mit einem Word-Embedding (der Schadenbeschreibungen) in einem neuronalen Netz berechnet werden kann. Die Datenaufbereitung soll sowohl für die Trainings- als auch für die Testdaten durchgeführt werden. Zeigen Sie für jeweils zwei geeignete Trainings- und Testdatensätze die ursprüngliche Schadenbeschreibung, die bereinige Schadenbeschreibung und die Zahlendarstellung als Token. Zeigen Sie an Beispielen, dass die Token in den Trainings- und Testdaten die gleiche Wortbedeutung haben.

Lösungsvorschlag:

Trainings- und Testdaten erzeugen:

```
# Datenaufteilung: 75% training, 25% test
n = nrow(dat)
trainIndex = sample(1:n, size = round(0.75*n), replace=FALSE)
train <- dat[trainIndex,]
test <- dat[-trainIndex,]
x_dtrain <- train$clean
y_dtrain <- train$line
x_dtest <- test$clean
y_dtest <- test$line
```

Tokenisierung, Sequenzen, Padding und Zielgröße vorbereiten:

```
# Vectorisierung der Token (Wörter/Wortteile)
max_features <- 100
tokenizer <- text_tokenizer(num_words = max_features) %>%
  fit_text_tokenizer(x_dtrain)

# Transform Text zu  Sequenz mit dem Training Tokenizer
seq_train <- texts_to_sequences(tokenizer, x_dtrain)
seq_test <- texts_to_sequences(tokenizer, x_dtest)

# Alle Sequenzen auf die gleiche Länge bringen
maxlen <- 10
x_train <- seq_train %>% pad_sequences(maxlen = maxlen)
x_test  <- seq_test  %>% pad_sequences(maxlen = maxlen)

# Die Sparte (Zielgröße) für Keras aufbereiten (to_categorial)
y_train_int <- as.integer(as.factor(y_dtrain))-1
y_test_int  <- as.integer(as.factor(y_dtest))-1
y_train <- to_categorical(y_train_int)
y_test  <- to_categorical(y_test_int)
```

Für die ersten Trainingsdatensätze die ursprüngliche Schadenbeschreibung, die bereinige Schadenbeschreibung und die Zahlendarstellung als Token anzeigen:

```
## Schadenbeschreibung (train): 1. Original
```

```
## [1] "water damage at Village Hall                      "
## [2] "hail/wind damage to roof of village hall          "
```

```
## Schadenbeschreibung (train): 2. Nach Bereinigung
```

```
## [1] "water damage Village Hall"
## [2] "hail / wind damage roof village hall"
```

```
## Schadenbeschreibung (train): 3. Darstellung als Token
```

```
## [[1]]
## [1]  5  1 49
##
## [[2]]
## [1] 25 11  1 24 49
```

Für dazu passende Testdatensätze die ursprüngliche Schadenbeschreibung, die bereinige Schadenbeschreibung und die Zahlendarstellung als Token anzeigen:

```
## Schadenbeschreibung (test): 1. Original
```

```
## [1] "wind and hail damage - distribution system        "
## [2] "lightning damaged a/c at City Hall                "
```

```
## Schadenbeschreibung (test): 2. Nach Bereinigung
```

```
## [1] "wind hail damage - distribution system"
## [2] "lightning damage / c City Hall"
```

```
## Schadenbeschreibung (test): 3. Darstellung als Token
```

```
## [[1]]
## [1] 11 25  1 46
##
## [[2]]
## [1]  3  1 87 49
```

Wortbedeutungen überprüfen: Token 1 (damage), Token 11 (wind), Token 25 (hail) und Token 49 (hall) haben in den Trainings- und Testdaten die gleiche Wortbedeutung. Je höher die Token-Nummer, desto seltener tritt das Wort auf. Daher spricht insbesondere die Gleichheit der weniger häufigen Token 25 und 49 dafür, dass die Token in den Training- und Testdaten die gleiche Wortbedeutung haben. Dies ist eine Voraussetzung für eine erfolgreiche Anwendung des zu entwickelnden, token-basierten Vorhersagemodells auf die Testdaten.

4. Spartenvorhersage mit einem neuronalen Netz und Word Embedding

Aufgabe B4: [Lernziel 5.1] [20 Punkte] Auf Basis der Schadenbeschreibungen soll die Sparte vorhergesagt werden (multinominale Klassifikation). Erstellen Sie hierfür mit Keras ein vorwärts gerichtetes, voll verknüpftes neuronales Netz (FC) mit einer 32-dimensionalen "Embedding Layer" und einer "Hidden Layer" mit 32 Neuronen. Zeigen Sie die Modellgröße

an (“summary”). Trainieren Sie das Modell (max. 20 epochs) mit Verwendung einer internen Validierungsstichprobe von 25%, zeigen Sie die Entwicklung von “Loss” und “Accuracy” über die Trainingsdurchläufe (epochs) an und bewerten Sie die Konvergenz. Erstellen Sie eine Sparten-Vorhersage anhand der Testdaten. Ermitteln und zeigen Sie die resultierende “Confusion Matrix” sowie Modellgenauigkeit und interpretieren Sie beides.

Lösungsvorschlag:

```
# Trainingsparameter für das neuronale Netz:
embedding_dims <- 32
hidden_dims <- 32
batch_size <- 32
epochs <- 10
```

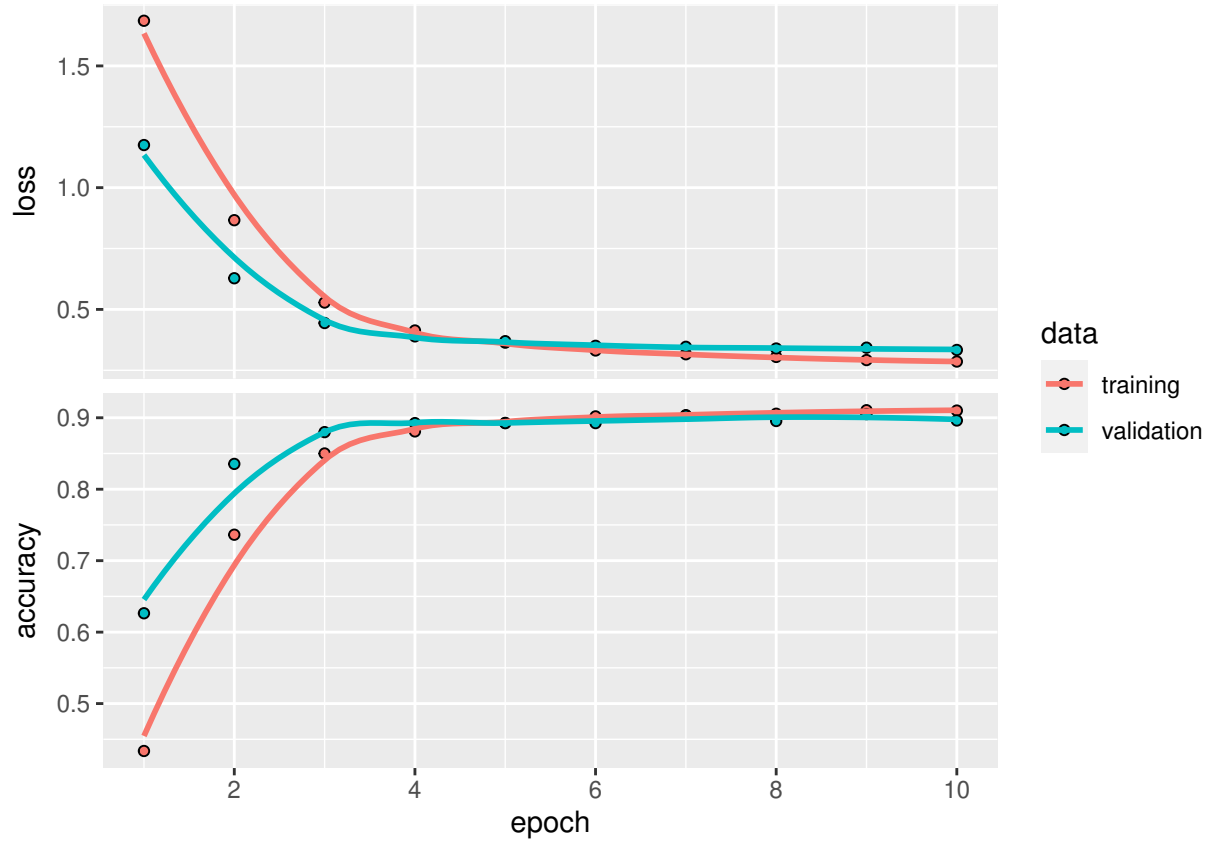
```
#Modell aufbauen
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features+1,
                  output_dim = embedding_dims,
                  input_length = maxlen) %>%
  layer_flatten() %>%
  layer_dense(units = hidden_dims, activation = "relu") %>%
  layer_dense(units = 8) %>%
  layer_activation("softmax") %>% compile(
    loss = "categorical_crossentropy",
    optimizer = "adam",
    metrics = "accuracy"
  )

# Modellgröße ausgeben
summary(model)
```

```
## Model: "sequential"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  embedding (Embedding)              (None, 10, 32)                  3232
##
##  flatten (Flatten)                  (None, 320)                     0
##
##  dense_1 (Dense)                    (None, 32)                      10272
##
##  dense (Dense)                      (None, 8)                       264
##
##  activation (Activation)            (None, 8)                       0
##
## =========================================================================
## Total params: 13,768
## Trainable params: 13,768
## Non-trainable params: 0
## _________________________________________________________________________
```

```
hist <- model %>%
  fit(
```

```
    x_train,
    y_train,
    batch_size = batch_size,
    epochs = epochs,
    validation_split = 0.25
  )
plot(hist)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Ergebnis: Das Modell konvergiert sehr schnell, bereits bei knapp 10 Durchläufen ist an der internen Validierungsstichprobe leichtes Overfitting zu beobachten.

Spartenvorhersage an Testdaten:

```
# Vorhersagen für Testdaten erzeugen
pred_test <- model %>% predict(x_test) %>% k_argmax()

# Confusion matrix erzeugen und anzeigen
cm <- confusionMatrix(factor(as.array(pred_test), levels=0:7), factor(y_test_int, levels=0:7))
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction   0   1   2   3   4   5   6   7
##          0  42   0   1   3   1   0   0   0
##          1   0  44   0   0   0   0   0   1
##          2   3   2 274   2   0   0   1   2
##          3   3   0   8  67  24   1  14   2
##          4  11   0   1  15 518   7   5   1
##          5   4   0   6  17  18 356   3   6
##          6   0   0   1   6   1   3 188   2
##          7   1   4   2   2   0   1  11 113
##
## Overall Statistics
##
##                Accuracy : 0.891
##                  95% CI : (0.8757, 0.905)
##     No Information Rate : 0.3126
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8647
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.65625  0.88000   0.9352  0.59821   0.9217   0.9674
## Specificity           0.99712  0.99943   0.9934  0.96916   0.9676   0.9622
## Pos Pred Value        0.89362  0.97778   0.9648  0.56303   0.9283   0.8683
## Neg Pred Value        0.98744  0.99658   0.9875  0.97320   0.9645   0.9914
## Prevalence            0.03560  0.02781   0.1630  0.06229   0.3126   0.2047
## Detection Rate        0.02336  0.02447   0.1524  0.03726   0.2881   0.1980
## Detection Prevalence  0.02614  0.02503   0.1580  0.06618   0.3103   0.2280
## Balanced Accuracy     0.82668  0.93971   0.9643  0.78369   0.9447   0.9648
##                      Class: 6 Class: 7
## Sensitivity            0.8468  0.88976
## Specificity            0.9918  0.98743
## Pos Pred Value         0.9353  0.84328
## Neg Pred Value         0.9787  0.99159
## Prevalence             0.1235  0.07063
## Detection Rate         0.1046  0.06285
## Detection Prevalence   0.1118  0.07453
## Balanced Accuracy      0.9193  0.93860
```

Interpretation: Das Modell sagt die Sparte an unbesehenen Testdaten in rund 89% der Fälle korrekt voraus (Accuracy). Die Sensitivität ist in der unspezifischen Sparte "Sonstiges" (Class 3, "Misc") auffällig - aber auch nachvollziehbar - gering.

5. Word Embedding analysieren

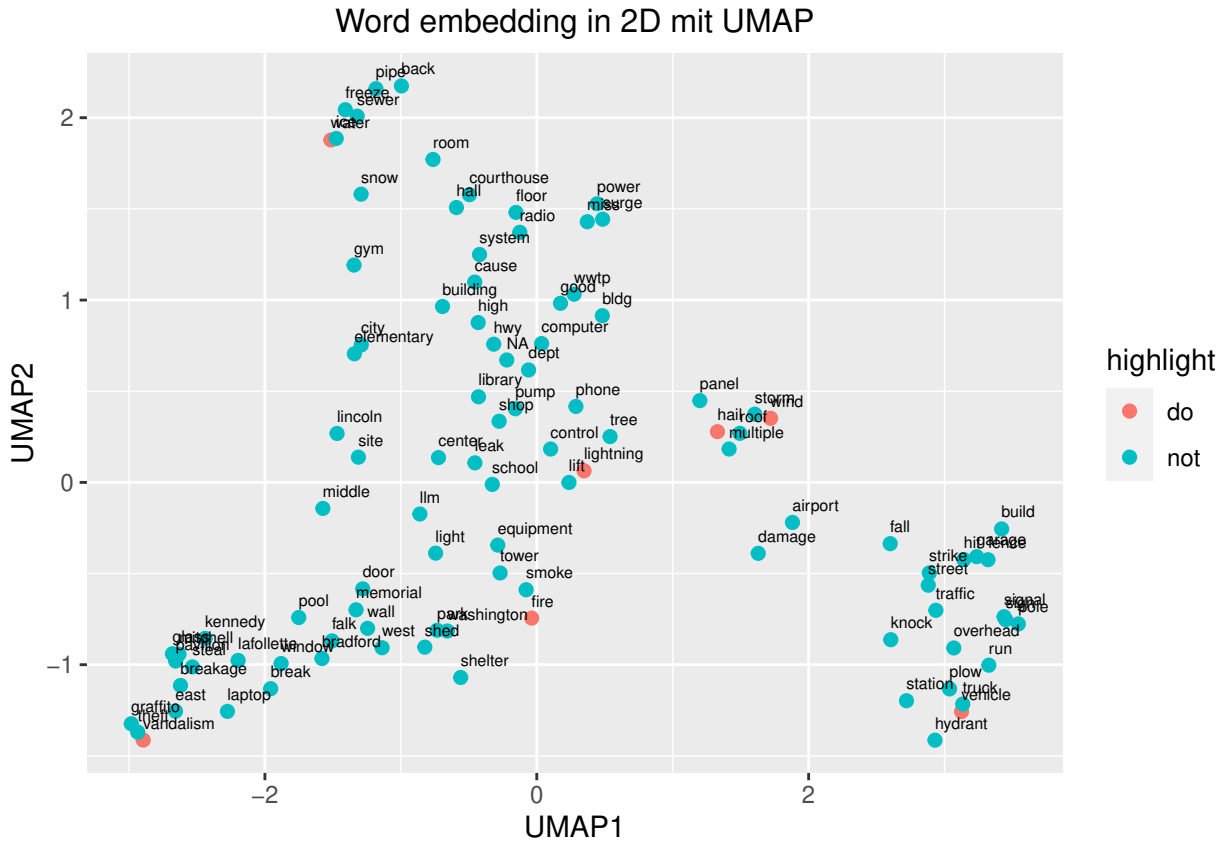
Aufgabe B5: [Lernziel 5.1] [15 Punkte] Erstellen Sie mit Hilfe der Dimensionreduktionsverfahrens UMAP oder t-SNE eine zweidimensionale Darstellung des hochdimensionalen Embeddings aus Aufgabe B4. Beschreiben und begründen Sie Ihr ausgewähltes Verfahren kurz und erläutern Sie die verwendeten Parameter. Erstellen Sie eine zweidimensionale graphische Darstellung des Ergebnisses der Dimensionsreduktion. Heben Sie dabei Worte, die

auch Spartennamen sind, farblich hervor. Kommentieren und bewerten Sie die dargestellte Wortverteilung im zweidimensionalen Raum. Berechnen Sie zum Vergleich die ähnlichsten Worte zu "water" sowie "storm" gemäß "cosine-similarity" der entsprechenden Wortvektoren und plausibilisieren Sie damit die graphische Darstellung.

Lösungsvorschlag:

t-SNE ("t-distributed stochastic neighbor embedding") und das recht neue UMAP ("Uniform Manifold Approximation and Projection") sind beides graphbasierte Methoden zur Dimensionsreduktion und Datenvisualisierung. Beide erzeugen einen hochdimensionalen Graphen und rekonstruieren ihn in einem weniger hochdimensionalen Raum unter Beibehaltung der Struktur, wobei UMAP generell weniger Rechenzeit benötigt sowie die globale Struktur besser erhält und daher hier bevorzugt wird. Wichtige UMAP-Parameter sind neben der Ausgabedimension (2) die Anzahl Nachbarn (15) sowie die Metrik (euklidisch), wobei hier die Standardwerte der R-Funktion uwot::umap verwendet werden.

```
# Embedding-Gewichte auslesen
emb_matrix <- get_weights(model)[[1]]

# UMAP Dimensionsreduktion durchführen
df_emb_umap <- as.data.frame(umap(emb_matrix, n_components = 2, metric = "euclidean",
                                  n_neighbors = 15), stringsAsFactors = FALSE)


# Klarnamen zuspielen
rownames(emb_matrix) <- c('NA',as.vector(unlist(tokenizer$index_word[1:100])))
df_emb_umap$word <- rownames(emb_matrix)
colnames(df_emb_umap) <- c("UMAP1", "UMAP2", "word")

# Besondere Worte markieren (für Plot)
marked <- list("fire","hail","lightning","vandalism", "vehicle", "water", "wind")
df_emb_umap$highlight <- ifelse(df_emb_umap$word %in% marked, 'do', 'not')

# Plot Word embedding mit Klarnamen
ggplot(df_emb_umap) +
  geom_point(aes(x = UMAP1, y = UMAP2, colour = highlight), size = 2) +
  geom_text(aes(UMAP1, UMAP2, label = word), size = 2, vjust=-1, hjust=0) +
  labs(title = "Word embedding in 2D mit UMAP") +
  theme(plot.title = element_text(hjust = .5, size = 12))
```



Kommentierung und Bewertung: “Water” ist von von anderen Aggregatzuständen ungeben, zu “vadalism” passt “graffito” und zu “fire” passt “smoke”, zu “vehicle” passt “truck” und zwischen “wind” und “hail” liegt “roof”, was von beiden Ereignissen beschädigt werden kann. Einige unspezifische Worte wie “middle” liegen weit von den näher spartenbezeichnenden Begriffen entfernt. Insgesamt macht die Darstellung einen plausiblen Eindruck.

Funktion zur Berechnung der ähnlichsten Wortvektoren gemäß cosine-similarity:

```
suppressPackageStartupMessages(library(text2vec))

words <- tibble(
  word = names(tokenizer$word_index),
  id = as.integer(unlist(tokenizer$word_index)))

words <- words %>%
  filter(id <= tokenizer$num_words) %>%
  arrange(id)

find_similar_words <- function(word, embedding_matrix) {
  similarities <- embedding_matrix[word, , drop = FALSE] %>%
    sim2(embedding_matrix, y = ., method = "cosine")

  similarities[,1] %>% sort(decreasing = TRUE) %>% head(5)
}
```

a) Die ähnlichsten Worte zu “water”:

```
find_similar_words("water", emb_matrix)
```

```
##     water       ice     sewer    freeze      snow
## 1.0000000 0.9447849 0.8493209 0.8384072 0.8137049
```

Die Wortähnlichkeiten von Wasser zu Eis, Kanal und Gefroren sind auch auf Deutsch gut nachvollziehbar und die UMAP stellt die hohen Ähnlichkeiten durch geringe Entfernungen hier gut dar.

b) Die ähnlichsten Worte zu “storm”:

```
find_similar_words("storm", emb_matrix)
```

```
##     storm      wind      roof  multiple      tree
## 1.0000000 0.7636139 0.6771144 0.5885508 0.5603752
```

Die Wortähnlichkeiten von Sturm zu “multiple”, Baum und Wind sind auch auf Deutsch gut nachvollziehbar und die UMAP stellt die Ähnlichkeiten durch geringe Entfernungen hier ebenfalls gut dar.

Ergebnis: In beiden Beispielen konnte die graphische Darstellung gut plausiblisiert werden.

```
# detach "text2vec" wg. namespace-Überschneidungen mit Keras
detach(package:text2vec, unload = TRUE)
```

6. Kreuzvalidierung und Embedding-Größe

Aufgabe B6: [Lernziel 6.1] [15 Punkte] Führen Sie auf Basis des in Aufgabe B4 entwickelten Vorhersagemodells für Embeddings mit 2, 4, 8, 16, 32 und 64 Dimensionen eine einheitliche, selbst programmierte fünffache Kreuzvalidierung für die sechs Modelle durch. Berechnen Sie die jeweilige Modellgenauigkeit (Accuracy) an der entsprechenden Validierungsstichprobe und emitteln Sie über die fünf Validierungsstichproben den Mittelwert der Modellgenauigkeit für jedes der sechs Modelle. Visualisieren Sie die Modellgenauigkeit mittels Boxplots. Zeigen Sie dazu auf der x-Achse die Embedding-Größe und auf der y-Achse die Modellgenauigkeit als Boxplot an und interpretieren Sie das Ergebnis.

Lösungsvorschlag:

Modellfunktion definieren:

```
build_model <- function(emb_dim) {
#Modell aufbauen
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features+1,
                  output_dim = emb_dim,
                  input_length = maxlen) %>%
  layer_flatten() %>%
  layer_dense(units = hidden_dims, activation = "relu") %>%
  layer_dense(units = 8)  %>%
  layer_activation("softmax") %>% compile(
    loss = "categorical_crossentropy",
    optimizer = "adam",
    metrics = "accuracy"
  )
}
```

fünffache Kreuzvalidierung für sechs Embedding-Dimensionierungen durchführen:

```
# k-fache Kreuzvalidierung
k <- 5
indices <- sample(1:nrow(train))
folds <- cut(1:length(indices), breaks = k, labels = FALSE)
num_epochs <- 10
# Ergebnisvektoren anlegen
acc_dim_2 <- c()
acc_dim_4 <- c()
acc_dim_8 <- c()
acc_dim_16 <- c()
acc_dim_32 <- c()
acc_dim_64 <- c()
# Startzeit merken
t <- proc.time()

for (i in 1:k) {
  cat("processing fold #", i, "\n")
  # Validierungsdaten vorbereiten: Daten der Partition # k
  val_indices <- which(folds == i, arr.ind = TRUE)
  val <- train[val_indices,]
  x_dval <- val$clean
  y_dval <- val$line
  # Trainingsdaten vorbereiten: Daten der anderen Partitionen
  dev <- train[-val_indices,]
  x_ddev <- dev$clean
  y_ddev <- dev$line

  # Transform Text zu  Sequenz mit dem Training Tokenizer
  seq_dev <- texts_to_sequences(tokenizer, x_ddev)
  seq_val <- texts_to_sequences(tokenizer, x_dval)
  # Alle Sequenzen auf die gleiche Länge bringen
  x_dev <- seq_dev %>% pad_sequences(maxlen = maxlen)
  x_val <- seq_val %>% pad_sequences(maxlen = maxlen)
  # Die Sparte (Zielgröße) für Keras aufbereiten (to_categorial)
  y_dev_int <- as.integer(as.factor(y_ddev))-1
  y_val_int <- as.integer(as.factor(y_dval))-1
  y_dev <- to_categorical(y_dev_int)
  y_val <- to_categorical(y_val_int)

  # Fit NN: 2-dim embedding
  model <- build_model(2)
  model %>% fit(x_dev, y_dev, epochs = num_epochs, batch_size = 32, verbose = 0)
  results <- model %>% evaluate(x_val, y_val, verbose = 0)
  acc_dim_2 <- c(acc_dim_2, getElement(results, 'accuracy'))
  # .. 4-dim embedding
  model <- build_model(4)
  model %>% fit(x_dev, y_dev, epochs = num_epochs, batch_size = 32, verbose = 0)
  results <- model %>% evaluate(x_val, y_val, verbose = 0)
  acc_dim_4 <- c(acc_dim_4, getElement(results, 'accuracy'))
  # .. 8-dim embedding
  model <- build_model(8)
  model %>% fit(x_dev, y_dev, epochs = num_epochs, batch_size = 32, verbose = 0)
```

```
  results <- model %>% evaluate(x_val, y_val, verbose = 0)
  acc_dim_8 <- c(acc_dim_8, getElement(results, 'accuracy'))
  # .. 16-dim embedding
  model <- build_model(16)
  model %>% fit(x_dev, y_dev, epochs = num_epochs, batch_size = 32, verbose = 0)
  results <- model %>% evaluate(x_val, y_val, verbose = 0)
  acc_dim_16 <- c(acc_dim_16, getElement(results, 'accuracy'))
  # .. 32-dim embedding
  model <- build_model(32)
  model %>% fit(x_dev, y_dev, epochs = num_epochs, batch_size = 32, verbose = 0)
  results <- model %>% evaluate(x_val, y_val, verbose = 0)
  acc_dim_32 <- c(acc_dim_32, getElement(results, 'accuracy'))
  # .. 64-dim embedding
  model <- build_model(64)
  model %>% fit(x_dev, y_dev, epochs = num_epochs, batch_size = 32, verbose = 0)
  results <- model %>% evaluate(x_val, y_val, verbose = 0)
  acc_dim_64 <- c(acc_dim_64, getElement(results, 'accuracy'))
}
```

```
## processing fold # 1
## processing fold # 2
## processing fold # 3
## processing fold # 4
## processing fold # 5
```

```
cat("Laufzeit Kreuzvalidierung", "\n")
```

```
## Laufzeit Kreuzvalidierung
```

```
proc.time()-t
```

```
##    user  system elapsed
##  165.79    5.12  133.00
```

```
cat("Modellgenauigkeit, Mittelwert:", "\n")
```

```
## Modellgenauigkeit, Mittelwert:
```

```
mean(acc_dim_2)
```

```
## [1] 0.8254258
```

```
mean(acc_dim_4)
```

```
## [1] 0.8771311
```

```
mean(acc_dim_8)
```

```
## [1] 0.8934413
```

```
mean(acc_dim_16)
```

```
## [1] 0.8943674
```

```
mean(acc_dim_32)
```

```
## [1] 0.8941813
```

```
mean(acc_dim_64)
```

```
## [1] 0.8954792
```

Daten für Boxplots aufbereiten:

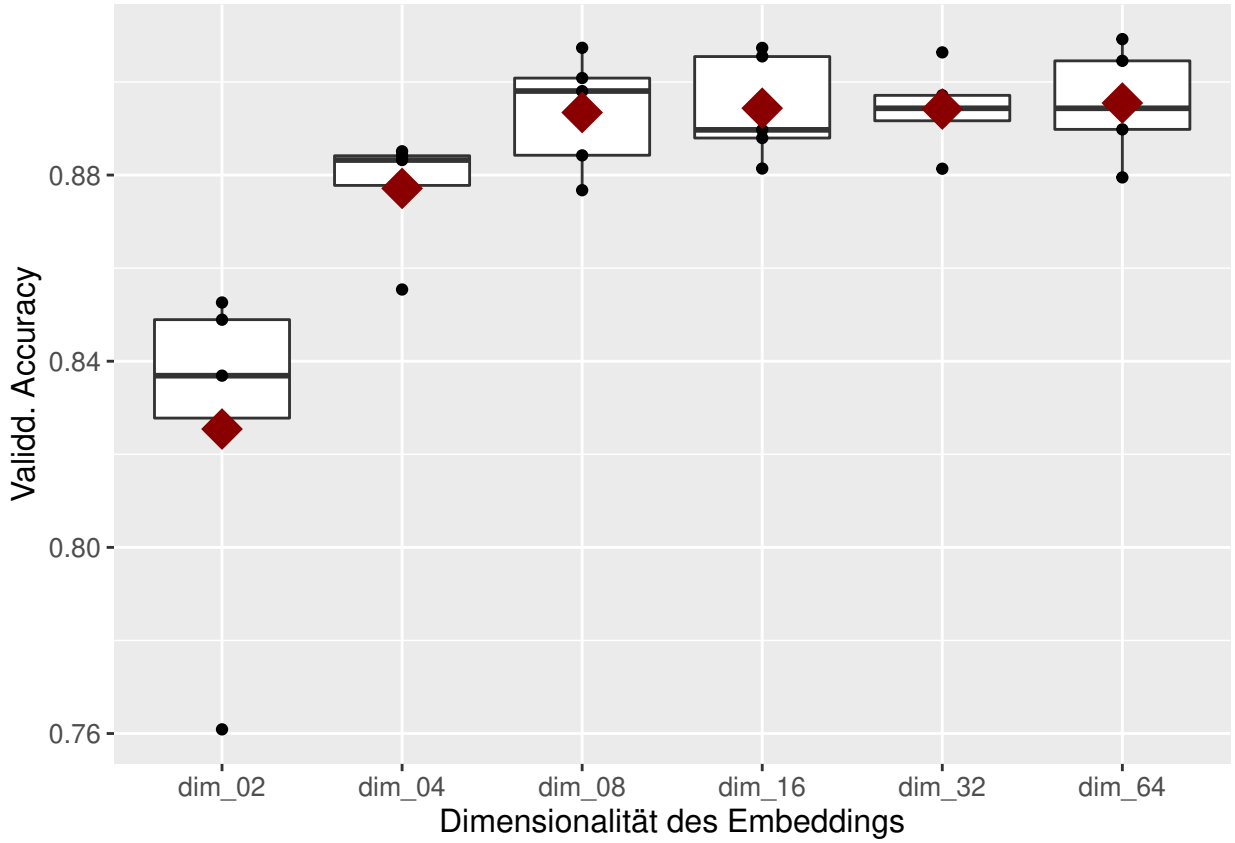
```
# write vectors in a Data Frame
df <- data.frame(1:5, acc_dim_2, acc_dim_4, acc_dim_8, acc_dim_16, acc_dim_32, acc_dim_64)
names(df) <- c("fold", "dim_02", "dim_04", "dim_08", "dim_16", "dim_32", "dim_64")
df
```

```
##   fold    dim_02    dim_04    dim_08    dim_16    dim_32    dim_64
## 1    1 0.8277778 0.8777778 0.8842593 0.8879629 0.8916667 0.8898148
## 2    2 0.8526413 0.8850788 0.9008341 0.9054680 0.9063948 0.9091752
## 3    3 0.7608897 0.8554217 0.8767377 0.8813716 0.8813716 0.8795181
## 4    4 0.8489342 0.8832252 0.9073216 0.9073216 0.8943466 0.9045413
## 5    5 0.8368860 0.8841520 0.8980538 0.8897127 0.8971270 0.8943466
```

```
# transform data from wide to long
df.long <- gather(df[1:7], size, Accuracy, dim_02, dim_04, dim_08, dim_16, dim_32, dim_64)
```

Boxplots erzeugen:

```
# create boxplot
ggplot(df.long, aes(size, Accuracy)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter(width = 0) +
  stat_summary(fun = mean, colour="darkred", geom="point",shape=18, size=7) +
#  coord_flip() +
  xlab("Dimensionalität des Embeddings") +
  ylab("Validd. Accuracy") +
  theme(text = element_text(size=12))
```



Interpretation: Mit zunehmender Embedding-Größe verbessert sich die Modellgenauigkeit bis acht Dimensionen deutlich. Bis 64 Dimensionen findet eine weitere (schwankende) leichte Verbesserung statt. Das Embedding sollte daher mindestens eine Diemsionalität von 8 aufweisen. Die Prognoseverbesserungen durch höherdimensionale Embeddings scheinen nicht signifikant zu sein. Das Modell mit dem 32-dimensionalen Embedding weist die geringsten Prognoseschwankungen auf und wird daher auch in der folgenden Aufgabe verwendet.

7. Spezielle neuronale Netze kreuzvaldiert anwenden

Aufgabe B7: [Lernziel 4.2] [20 Punkte] a) Ziel dieser Aufgabe ist festzustellen, ob durch rekurrente neuronale Netze (RNN) und andere tiefe spezielle neuronale Netze wie Faltungsnetze (ConvNet) das Prognoseergebnis verbessert werden kann. Erläutern Sie, weshalb hier RNNs und ConvNets bzw. alternativ zu ConvNets ein von Ihnen bevorzugtes neueres Deep-Learning-Verfahren zu einer Prognoseverbesserung führen könnten. b) Zur technischen Umsetzung können in Aufgabe B6 entwickelte Programmteile verwendet bzw. adaptiert werden. Die fünffache Kreuzvalidierung soll beibehalten werden. Es soll wie bisher Keras genutzt und in jedem neuronalen Netz eingangs ein Word Embedding verwendet werden, nun einheitlich mit 32 Dimensionen. Anstelle mehrerer neuronalen Netze mit unterschiedlichen Embedding-Größen (Aufgabe B6) sollen nun mehrere RNNs mit unterschiedlicher Anzahl an "units" (Größe des Ausgaberaums) berechnet werden. Konkret soll dazu im Vergleich zu Aufgabe B6 (mindestens) die hidden layer entfernt und durch LSTM-Layer mit 8, 16 bzw. 32 units (drei Modelle) ersetzt werden. Zusätzlich soll ein anderes tiefes spezielles neuronales Netz, beispielsweise mit einer eindimensionalen Faltungsschicht (conv-1d-layer) geeignet mit in die Betrachtung einbezogen werden. Analog Aufgabe B6 soll die jeweilige Modellgenauigkeit (Accuracy) an der entsprechenden Validierungsstichprobe berechnet werden und über die fünf Validierungsstichproben den Mittelwert der Modellgenauigkeit für jedes der vier Modelle ermittelt werden. c)

Visualisieren Sie die Prognosegenauigkeit der vier neuen Modelle analog Aufgabe B6 mittels Boxplots. Nehmen Sie zusätzlich die entsprechenden Werte des (voll verknüpften vorwärts gerichteten) FC-Modells mit 32-dimensionalem Embedding aus Aufgabe B6 mit in den Boxplot auf und interpretieren Sie die Genauigkeit samt Schwankungen. d) Berechnen Sie für die jeweilige Validierungsstichprobe die Differenz der Prognosegenauigkeit der einzelnen vier neuen Modelle mit dem oben genannten FC-Modell und visualisieren Sie die Differenzen in einen weiteren Boxplot für die vier neuen Modelle. Erläutern Sie, was durch die Differenzbildung erreicht wurde und interpretieren sie das Ergebnis.

Lösungsvorschlag:

a) RNNs und eindimensionale Faltungsnetze (bzw. layer) eignen sich gut für Zeitreihen und Sequenzen, da sie im Gegensatz zu den bislang verwendeten voll verknüpften vorwärts gerichteten neuronalen Netzen die "Umgebungsinformation" wie Wortreihenfolge oder lokale Muster berücksichtigen und daher hier potentiell zu einer Prognoseverbesserung führen können. Aufgrund der kurzen Sequenzen und bereits beobachteter lokaler Schlüsselworte/-phrasen könnten ConvNets hier noch besser als RNNs/LSTMs funktionieren.

b) Technische Umsetzung der Modelle und Prognosen

Modellfunktion für LSTM und eindimensionales ConvNet definieren:

```
build_lstm <- function(lstm_units) {
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features+1,
                  output_dim = 32,
                  input_length = maxlen) %>%
  layer_lstm(units = lstm_units) %>%
  layer_dense(units = 8)  %>%
  layer_activation("softmax") %>% compile(
    loss = "categorical_crossentropy",
    optimizer = "adam",
    metrics = "accuracy"
  )
}

build_cnn <- function(filter_n) {
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features+1,
                  output_dim = 32,
                  input_length = maxlen) %>%
  layer_dropout(0.2) %>%
  layer_conv_1d(filters = filter_n, kernel_size = 3,
    padding = "valid", activation = "relu", strides = 1) %>%
  layer_global_max_pooling_1d() %>%
  layer_dense(hidden_dims) %>%
  layer_dropout(0.2) %>%
  layer_activation("relu") %>%#  layer_flatten() %>%
#  layer_dense(units = hidden_dims, activation = "relu") %>%
  layer_dense(units = 8) %>%
  layer_activation("softmax") %>% compile(
    loss = "categorical_crossentropy",
    optimizer = "adam",
    metrics = "accuracy"
```

```
  )
}
```

fünffache Kreuzvalidierung für vier Embedding-Dimensionierungen durchführen:

```
# k-fache Kreuzvalidierung
k <- 5
indices <- sample(1:nrow(train))
folds <- cut(1:length(indices), breaks = k, labels = FALSE)
num_epochs <- 10
# Ergebnisvektoren anlegen
acc_lstm_8 <- c()
acc_lstm_16 <- c()
acc_lstm_32 <- c()
acc_cnn_32f <- c()
# Startzeit merken
t <- proc.time()

for (i in 1:k) {
  cat("processing fold #", i, "\n")
  # Validierungsdaten vorbereiten: Daten der Partition # k
  val_indices <- which(folds == i, arr.ind = TRUE)
  val <- train[val_indices,]
  x_dval <- val$clean
  y_dval <- val$line
  # Trainingsdaten vorbereiten: Daten der anderen Partitionen
  dev <- train[-val_indices,]
  x_ddev <- dev$clean
  y_ddev <- dev$line

  # Transform Text zu  Sequenz mit dem Training Tokenizer
  seq_dev <- texts_to_sequences(tokenizer, x_ddev)
  seq_val <- texts_to_sequences(tokenizer, x_dval)
  # Alle Sequenzen auf die gleiche Länge bringen
  x_dev <- seq_dev %>% pad_sequences(maxlen = maxlen)
  x_val <- seq_val %>% pad_sequences(maxlen = maxlen)
  # Die Sparte (Zielgröße) für Keras aufbereiten (to_categorial)
  y_dev_int <- as.integer(as.factor(y_ddev))-1
  y_val_int <- as.integer(as.factor(y_dval))-1
  y_dev <- to_categorical(y_dev_int)
  y_val <- to_categorical(y_val_int)

  # .. lstm1
  model <- build_lstm(8)
  model %>% fit(x_dev, y_dev, epochs = num_epochs, batch_size = 32, verbose = 0)
  results <- model %>% evaluate(x_val, y_val, verbose = 0)
  acc_lstm_8 <- c(acc_lstm_8, getElement(results, 'accuracy'))
  # .. lstm2
  model <- build_lstm(16)
  model %>% fit(x_dev, y_dev, epochs = num_epochs, batch_size = 32, verbose = 0)
  results <- model %>% evaluate(x_val, y_val, verbose = 0)
  acc_lstm_16 <- c(acc_lstm_16, getElement(results, 'accuracy'))
  # .. lstm3
```

```
  model <- build_lstm(32)
  model %>% fit(x_dev, y_dev, epochs = num_epochs, batch_size = 32, verbose = 0)
  results <- model %>% evaluate(x_val, y_val, verbose = 0)
  acc_lstm_32 <- c(acc_lstm_32, getElement(results, 'accuracy'))
  # .. cnn
  model <- build_cnn(32)
  model %>% fit(x_dev, y_dev, epochs = num_epochs, batch_size = 32, verbose = 0)
  results <- model %>% evaluate(x_val, y_val, verbose = 0)
  acc_cnn_32f <- c(acc_cnn_32f, getElement(results, 'accuracy'))
}
```

```
## processing fold # 1
## processing fold # 2
## processing fold # 3
## processing fold # 4
## processing fold # 5
```

```
cat("Laufzeit Kreuzvalidierung", "\n")
```

```
## Laufzeit Kreuzvalidierung
```

```
proc.time()-t
```

```
##    user  system elapsed
##  337.13    9.55  199.97
```

```
cat("Modellgenauigkeit, Mittelwert:", "\n")
```

```
## Modellgenauigkeit, Mittelwert:
```

```
mean(acc_lstm_8)
```

```
## [1] 0.8962206
```

```
mean(acc_lstm_16)
```

```
## [1] 0.8978888
```

```
mean(acc_lstm_32)
```

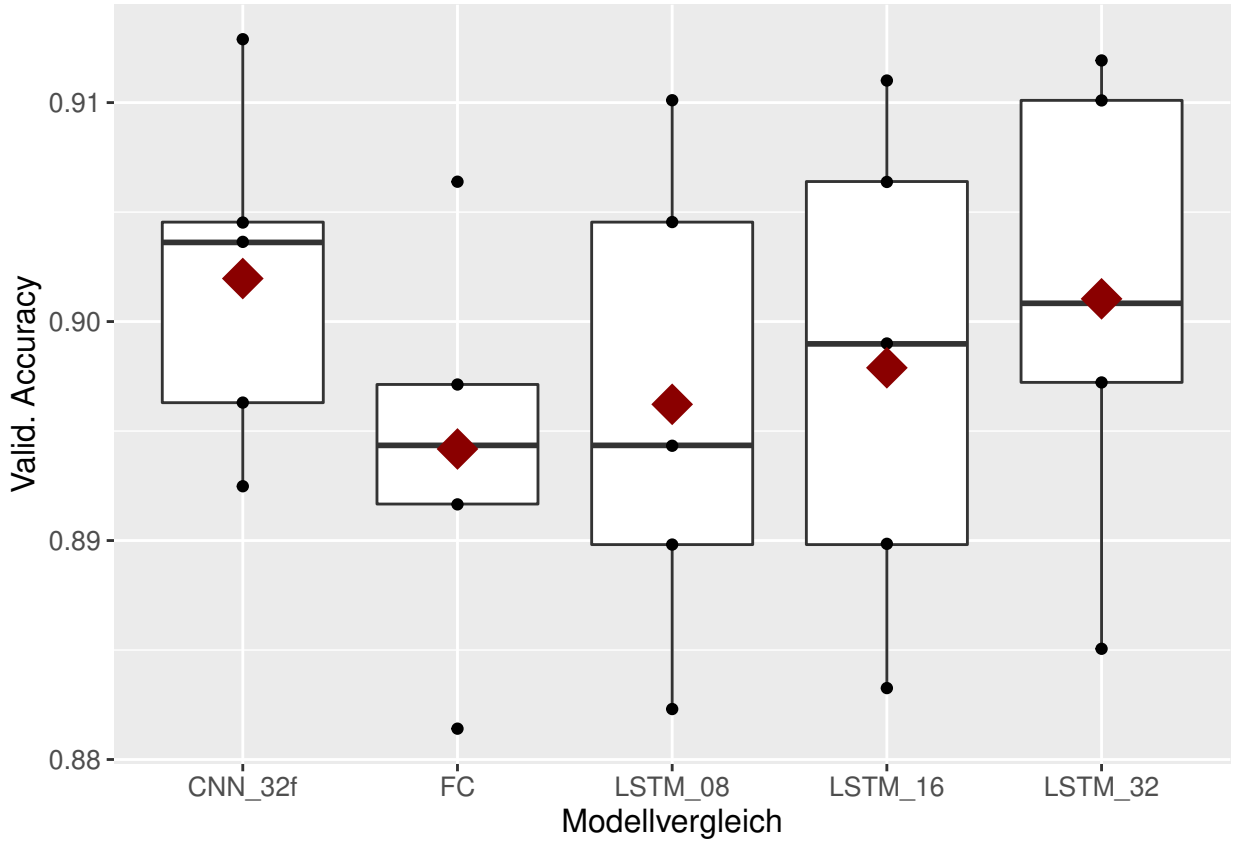
```
## [1] 0.9010385
```

```
mean(acc_cnn_32f)
```

```
## [1] 0.9019655
```

c) Prognosegenauigkeit: Daten für Boxplots aufbereiten und Boxplots erzeugen

```r
# Vektoren in Data-Frame schreiben
df <- data.frame(1:5, acc_dim_32, acc_lstm_8, acc_lstm_16, acc_lstm_32, acc_cnn_32f)
names(df) <- c("fold", "FC", "LSTM_08", "LSTM_16", "LSTM_32", "CNN_32f")
# Daten pivotieren (von breit auf lang)
df.long <- gather(df[1:6], size, Accuracy, FC, LSTM_08, LSTM_16, LSTM_32, CNN_32f)
# Boxplot erzeugen
ggplot(df.long, aes(size, Accuracy)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter(width = 0) +
  stat_summary(fun = mean, colour="darkred", geom="point",shape=18, size=7) +
  xlab("Modellvergleich") +
  ylab("Valid. Accuracy") +
  theme(text = element_text(size=12))
```

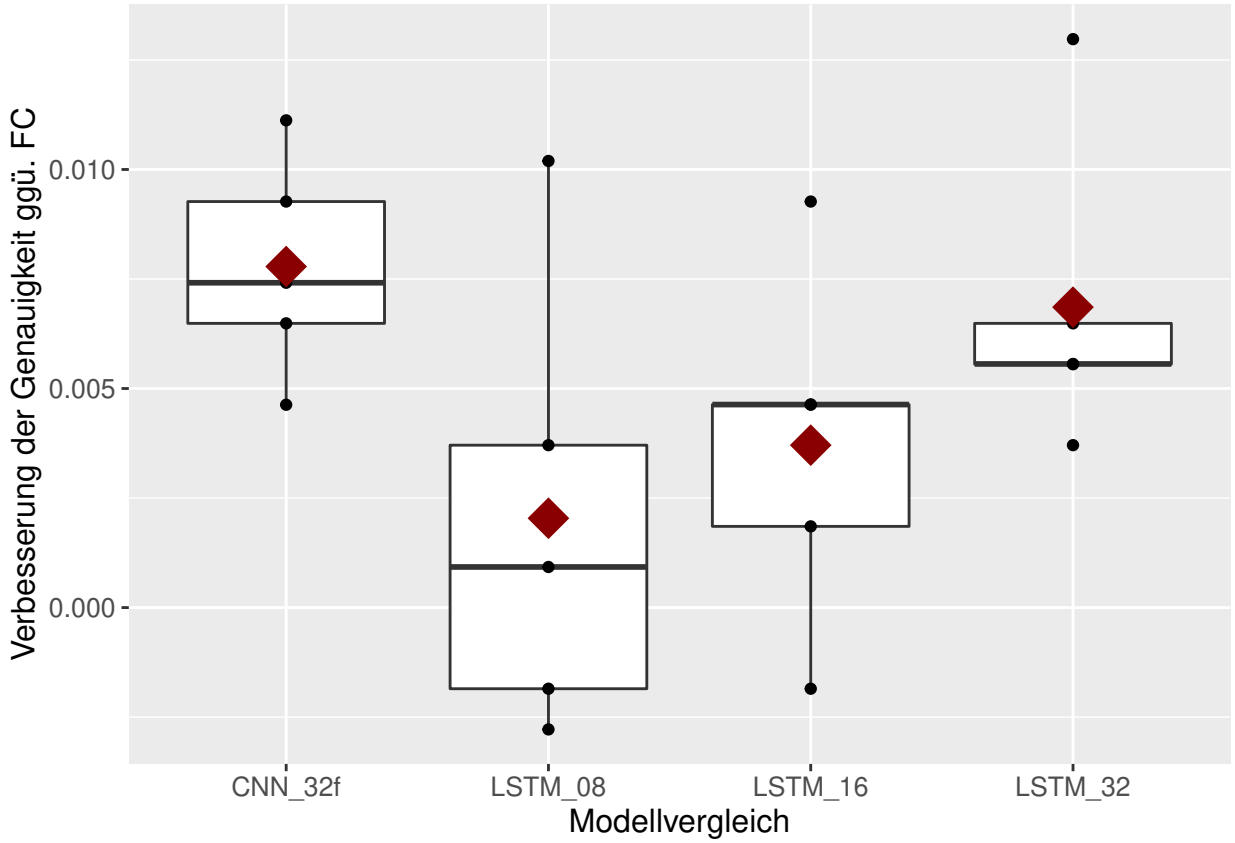


Interpretation: Die Mittelwerte der Prognosegenauigkeit der LSTMs und CNNs sind etwas höher als die des einfacheren FC-Modells, die Prognosegenauigkeit liegt erstmals leicht über 90%. Allerdings schwankt die Prognosegenaugkeit der neuen Modelle stärker. Möglicherweise sind die neuen Modelle bereits zu komplex und instabil. Insgesamt kann aus dieser Darstellung keine signifikante Verbesserung abgeleitet werden.

d) Prognoseverbesserung durch die neuen, speziellen neuronalen Netze ggü. FC-Modell

```r
# Vektoren in Data-Frame schreiben
df <- data.frame(1:5, acc_dim_32, acc_lstm_8, acc_lstm_16, acc_lstm_32, acc_cnn_32f)
names(df) <- c("fold", "FC", "LSTM_08", "LSTM_16", "LSTM_32", "CNN_32f")
# Prognoseverbesserung ggü. FC-Modell berechnen
df$LSTM_08 <- df$LSTM_08 - df$FC
```

```r
df$LSTM_16 <- df$LSTM_16 - df$FC
df$LSTM_32 <- df$LSTM_32 - df$FC
df$CNN_32f <- df$CNN_32f - df$FC
# Daten pivotieren (von breit auf lang)
df.long <- gather(df[1:6], size, Accuracy_Diff, LSTM_08, LSTM_16, LSTM_32, CNN_32f)
# Boxplot erzeugen
ggplot(df.long, aes(size, Accuracy_Diff)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter(width = 0) +
  stat_summary(fun = mean, colour="darkred", geom="point",shape=18, size=7) +
  xlab("Modellvergleich") +
  ylab("Verbesserung der Genauigkeit ggü. FC") +
  theme(text = element_text(size=12))
```



Interpretation: Durch die Differenzbildung konnte der Einfluss der alle Modelle betreffenden Varianz zwischen den Stichproben (können leicht unterschiedlich ausfallen, da ohne Stratifizierung) neutralisiert werden. Sowohl die Prognoseverbesserung des LSTM-Modell mit 32 units als auch die des CNN (filters: 32 convolutions) ist nun klar signifikant. Insgesamt kann durch die Anwendung der tiefen speziellen neuronalen Netze LSTM und 1D-ConvNet hier eine Prognoseverbesserung von knapp einem Prozent ohne wesentliche Nachteile bei der Laufzeit erzielt werden.

```r
# Anhang: Verwendete Versionen
sessionInfo()
```

```
## R version 4.2.0 (2022-04-22 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
```

```
## Running under: Windows 10 x64 (build 19044)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=German_Germany.utf8  LC_CTYPE=German_Germany.utf8
## [3] LC_MONETARY=German_Germany.utf8 LC_NUMERIC=C
## [5] LC_TIME=German_Germany.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] keras_2.8.0          tensorflow_2.8.0     uwot_0.1.11
##  [4] Matrix_1.4-1         caret_6.0-92         lattice_0.20-45
##  [7] forcats_0.5.1        stringr_1.4.0        dplyr_1.0.9
## [10] purrr_0.3.4          readr_2.1.2          tidyr_1.2.0
## [13] tibble_3.1.7         ggplot2_3.3.6        tidyverse_1.3.1
## [16] tidytext_0.3.2       tm_0.7-8             NLP_0.2-1
## [19] textstem_0.1.4       koRpus.lang.en_0.1-4 koRpus_0.13-8
## [22] sylly_0.1-6
##
## loaded via a namespace (and not attached):
##   [1] readxl_1.4.0           backports_1.4.1        plyr_1.8.7
##   [4] splines_4.2.0          listenv_0.8.0          tfruns_1.5.0
##   [7] SnowballC_0.7.0        digest_0.6.29          foreach_1.5.2
##  [10] htmltools_0.5.2        float_0.3-0            fansi_1.0.3
##  [13] magrittr_2.0.3         tzdb_0.3.0             recipes_0.2.0
##  [16] globals_0.15.0         modelr_0.1.8           gower_1.0.0
##  [19] textshape_1.7.3        hardhat_0.2.0          colorspace_2.0-3
##  [22] rvest_1.0.2            rappdirs_0.3.3         haven_2.5.0
##  [25] xfun_0.30              crayon_1.5.1           jsonlite_1.8.0
##  [28] zeallot_0.1.0          survival_3.3-1         iterators_1.0.14
##  [31] glue_1.6.2             gtable_0.3.0           ipred_0.9-12
##  [34] future.apply_1.9.0     mlapi_0.1.1            scales_1.2.0
##  [37] DBI_1.1.2              qdapRegex_0.7.5        Rcpp_1.0.8.3
##  [40] reticulate_1.24        proxy_0.4-26           textclean_0.9.3
##  [43] stats4_4.2.0           lava_1.6.10            prodlim_2019.11.13
##  [46] httr_1.4.3             FNN_1.1.3              ellipsis_0.3.2
##  [49] pkgconfig_2.0.3        farver_2.1.0           nnet_7.3-17
##  [52] dbplyr_2.1.1           utf8_1.2.2             here_1.0.1
##  [55] tidyselect_1.1.2       labeling_0.4.2         rlang_1.0.2
##  [58] reshape2_1.4.4         munsell_0.5.0          cellranger_1.1.0
##  [61] tools_4.2.0            cli_3.3.0              generics_0.1.2
##  [64] rsparse_0.5.0          broom_0.8.0            evaluate_0.15
##  [67] fastmap_1.1.0          yaml_2.3.5             RhpcBLASctl_0.21-247.1
##  [70] ModelMetrics_1.2.2.2   knitr_1.39             fs_1.5.2
##  [73] lgr_0.4.3              future_1.25.0          nlme_3.1-157
##  [76] whisker_0.4            slam_0.1-50            xml2_1.3.3
##  [79] tokenizers_0.2.1       compiler_4.2.0         rstudioapi_0.13
##  [82] png_0.1-7              e1071_1.7-9            reprex_2.0.1
##  [85] syuzhet_1.0.6          stringi_1.7.6          highr_0.9
##  [88] RSpectra_0.16-1        vctrs_0.4.1            pillar_1.7.0
##  [91] lifecycle_1.0.1        data.table_1.14.2      sylly.en_0.1-3
```

```
##  [94] R6_2.5.1              janeaustenr_0.1.5     parallelly_1.31.1
##  [97] lexicon_1.2.1         codetools_0.2-18      MASS_7.3-56
## [100] assertthat_0.2.1      rprojroot_2.0.3       withr_2.5.0
## [103] mgcv_1.8-40           parallel_4.2.0        hms_1.1.1
## [106] grid_4.2.0            rpart_4.1.16          timeDate_3043.102
## [109] class_7.3-20          rmarkdown_2.14        pROC_1.18.0
## [112] lubridate_1.8.0       base64enc_0.1-3
```