Modell erstellen mit Hilfe eines künstlichen neuronalen Netzes

- Phase: Data Modelling -

In diesem Notebook wird ein Neuronales Netz erzeugt um die Ergebnisse vorherzusagen. Ziel ist eine möglichst hohe Trefferquote zu erzielen.

Dieses Notebook nutzt die Dateien training.pkl und test.pkl.

In diesem Notebook werden keine weiteren Dateien erzeugt.

Zunächst wird die Art des neuronalen Netzes definiert, welches verwendet werden soll.

Die Implementierung erfolgt mittels der Python Deep Learning Bibliothek Keras. Keras ist eine Bibliothek, welche benutzerorientiert designt wurde. (vergl. https://keras.io/ (https://keras.io/))

In [1]:
```
from keras.models import Sequential
from keras.layers import Dense
#from sklearn.model_selection import train_test_split # Wird nicht benötigt...
import pandas as pd
import numpy as np
```

In [2]:
```
DATA_PATH = '../data/'
```

Einlesen der vorbereiteten Test- und Trainingsdaten

In [3]:
```
df_train = pd.read_pickle(DATA_PATH+'training.pkl')
df_test =  pd.read_pickle(DATA_PATH+'test.pkl')
```

In [4]:
```
df_train.head()
```

Out[4]:

	date	count_in	count_out	day_of_week	dewpoint	humidity	precipitation	pressure	temperature	winddirection	...	hour_19	hour_20	h
0	2015-01-01	42	54	1	-14.0	29.0	0.0	1026.7	2.2	220.0	...	0	0	
23	2015-01-01	36	32	1	-9.0	33.0	0.0	1018.8	6.1	210.0	...	0	0	
22	2015-01-01	57	54	1	-9.2	30.0	0.0	1019.0	7.2	210.0	...	0	0	
21	2015-01-01	74	71	1	-9.2	30.0	0.0	1019.1	7.2	220.0	...	0	0	
20	2015-01-01	93	90	1	-10.1	27.0	0.0	1019.5	7.8	240.0	...	0	1	

5 rows × 45 columns

In [5]:
```
df_train.shape
```

Out[5]:
```
(17539, 45)
```

Training und Vorhersage von zwei Rückgabewerte durch das Neuronale Netz

Dieses Kapitel beschreibt die Erstellung und das Testing eines Neuronalen Netzes mit den beiden Zielvariablen Count_in und Count_out also der Anzahl der Ausleihe- und Rückgabevorgänge.

Die Eingabedaten für das künstliche Neuronale Netz werden zunnächst in das korrekte Format umgewandelt um mit der Programmiersprache und der Bibliothek Keras arbeiten zu können. Hintergrund ist, dass die Performance extrem langsam ist, wenn als Eingabe ein Pandas Dataframe verwendet wird. (Orginalzitat: "Directly using Pandas in a neural network would be absolutely ridiculous. The performance would be abysmal." (Quelle: https://stackoverflow.com/questions/59892913/pandas-data-frame-used-as-input-for-neural-network (https://stackoverflow.com/questions/59892913/pandas-data-frame-used-as-input-for-neural-network))

Daher werden die Inputdaten, welche in Form eines Pandas Dataframe vorliegen, zunächst in ein Numpy Array umgewandelt.

In [6]:
```
cols = df_train.columns.tolist()
```

In [7]:
```
# Die Werte entfernen, welche vorhergesagt werden sollen: Count_in und Count_out
cols.remove('count_in')
cols.remove('count_out')
cols
```

Out[7]:
```
['date',
 'day_of_week',
 'dewpoint',
 'humidity',
 'precipitation',
 'pressure',
 'temperature',
 'winddirection',
 'windspeed',
 'workingday',
 'year',
 'Season_Fall',
 'Season_Spring',
 'Season_Summer',
 'hour_0',
 'hour_1',
 'hour_2',
 'hour_3',
 'hour_4',
 'hour_5',
 'hour_6',
 'hour_7',
 'hour_8',
 'hour_9',
 'hour_10',
 'hour_11',
 'hour_12',
 'hour_13',
 'hour_14',
 'hour_15',
 'hour_16',
 'hour_17',
 'hour_18',
 'hour_19',
 'hour_20',
 'hour_21',
 'hour_22',
 'weekday_Friday',
 'weekday_Monday',
 'weekday_Saturday',
 'weekday_Thursday',
 'weekday_Tuesday',
 'weekday_Wednesday']
```

Die Zielwerte des neuronalen Netzes definieren, da das Neuronale Netz zwei Werte vorhersagen soll, werden die Attribute count_in und count_out als Zielwerte definiert.

In [8]:
```
y_train = df_train[['count_in','count_out']].values
```

Die Attribute festlegen, welche für das Training berücksichtigt werden sollen. In diesem Fall alle Attribute mit Ausnahme von count_out und count_in also der Anzahl der Ausleihe- und Rückgabevorgänge.

In [9]:
```
X_train = df_train[cols].values
```

Nachdem die Attribute zugeordnet wurden, werden die Werte nun noch in den Datentyp float umgewandelt, damit sie als Input für das künstliche Neuronale Netzwerk verwendet werden können.

In [11]:
```
X_train = np.asarray(X_train).astype(np.float32)
```

In [12]:
```
y_train = np.asarray(y_train).astype(np.float32)
```

Aufbau eines künstlichen neuronalen Netzes:

Ein neuronales Netz besteht aus mindestens drei Schichten. Aus genau einer Eingabeschicht, mindestens einer verdeckten Schicht und exakt einer Ausgabeschicht. Jede Schicht kann hierbei eine verschiedene Anzahl von "Neuronen" beinhalten solange jedes Neuron mit einem Vorgänger und einem Nachfolger verbunden ist.

Um das Modell des Neuronalen Netzes zu erstellen wird die Methode Sequential aus der Bibliothek Keras verwendet.

In [15]:
```
#Modell erstellen.
model = Sequential()
```

Das künstliche neuronale Netz wird mit 43 Eingabeneuronen in der Eingabeschicht erstellt. Dies entspricht der Anzahl der Attribute im Trainingsdatensatz. Der Hidden Layer wird mit 100 Neuronen in der ersten Hidden Layer Schicht definiert.

In [16]:
```
# Hidden Layer(100 Neuronen), Input Layer (43 Neuronen)
model.add(Dense(100, input_dim=43, kernel_initializer='normal', activation='relu'))
```

Die Ausgabeschicht besteht aus zwei Neuronen, da genau zwei Attribute vorhergesagt werden sollen.

In [17]:
```
# Das Output Layer hat zwei Neuronen
model.add(Dense(2, kernel_initializer='normal',activation='sigmoid'))
```

In den nächsten beiden Schritten werden die Optimierungs- und Verlustfunktion (sog. Lossfunktion) festgelegt sowie das Modell mit den Trainingsdaten trainiert.

In [18]:
```
# Optimierungs- und Lossfunktion festlegen
model.compile(optimizer='rmsprop', loss='binary_crossentropy',metrics=['accuracy'])
```

In [19]:
```
# Training des Modells
model.fit(X_train, y_train, epochs=10, batch_size=25)
```

```
Epoch 1/10
702/702 [==============================] - 0s 671us/step - loss: nan - accuracy: 0.5489
Epoch 2/10
702/702 [==============================] - 0s 641us/step - loss: nan - accuracy: 0.5489
Epoch 3/10
702/702 [==============================] - 0s 633us/step - loss: nan - accuracy: 0.5489
Epoch 4/10
702/702 [==============================] - 0s 670us/step - loss: nan - accuracy: 0.5489
Epoch 5/10
702/702 [==============================] - 0s 631us/step - loss: nan - accuracy: 0.5489
Epoch 6/10
702/702 [==============================] - 0s 650us/step - loss: nan - accuracy: 0.5489
Epoch 7/10
702/702 [==============================] - 0s 650us/step - loss: nan - accuracy: 0.5489
Epoch 8/10
702/702 [==============================] - 0s 644us/step - loss: nan - accuracy: 0.5489
Epoch 9/10
702/702 [==============================] - 1s 872us/step - loss: nan - accuracy: 0.5489
Epoch 10/10
702/702 [==============================] - 1s 900us/step - loss: nan - accuracy: 0.5489
```

Out[19]:
```
<tensorflow.python.keras.callbacks.History at 0x287a5deea48>
```

Validierung des Modells auf den Testdaten.

Zunächst werden die Testdaten für das Modell vorbereitet. Da künstliche Neuronale Netze "nur" mit Gleitkommazahlen umgehen können werden die Werte hier vorbereitet.

In [22]:
```
# Vorbereitung der Testdaten
y_test = df_test[['count_in','count_out']].values
```

In [23]:
```
X_test = df_test[cols].values
```

In [24]:
```
# Werte in Floats umwandeln
y_test = np.asarray(y_test).astype(np.float32)
```

In [25]:
```
X_test = np.asarray(X_test).astype(np.float32)
```

In [26]:
```
# Evaluation des Modells mit den Testdaten
score = model.evaluate(X_test, y_test)
print("Accuracy Score: "+str(round(score[1],4)))
```

```
272/272 [==============================] - 0s 691us/step - loss: nan - accuracy: 0.5299
Accuracy Score: 0.5299
```

Training und Vorhersage eines Rückgabewertes durch das Neuronale Netz

In diesem Kapitel soll die Anzahl der Ausleihvorgänge ("count_out") vorhergesagt werden.

In [19]:
```
colssingle = df_train.columns.tolist()
```

In [20]:
```
colssingle.remove('count_out')
colssingle.remove('count_in')
colssingle
```

Out[20]:
```
['date',
 'day_of_week',
 'dewpoint',
 'humidity',
 'precipitation',
 'pressure',
 'temperature',
 'winddirection',
 'windspeed',
 'workingday',
 'year',
 'Season_Fall',
 'Season_Spring',
 'Season_Summer',
 'hour_0',
 'hour_1',
 'hour_2',
 'hour_3',
 'hour_4',
 'hour_5',
 'hour_6',
 'hour_7',
 'hour_8',
 'hour_9',
 'hour_10',
 'hour_11',
 'hour_12',
 'hour_13',
 'hour_14',
 'hour_15',
 'hour_16',
 'hour_17',
 'hour_18',
 'hour_19',
 'hour_20',
 'hour_21',
 'hour_22',
 'weekday_Friday',
 'weekday_Monday',
 'weekday_Saturday',
 'weekday_Thursday',
 'weekday_Tuesday',
 'weekday_Wednesday']
```

In [21]:
```
y_trainsingle = df_train[['count_out']].values
```

In [22]:
```
X_trainsingle = df_train[colssingle].values
```

In [23]:
```
X_trainsingle = np.asarray(X_trainsingle).astype(np.float32)
```

In [24]:
```
y_trainsingle = np.asarray(y_trainsingle).astype(np.float32)
```

In [31]:
```
#Modell mit einem Zielwert erstellen.
singlemodel = Sequential()
```

In [32]:
```
# Hidden Layer(100 Neuronen), Input Layer (43 Neuronen)
singlemodel.add(Dense(100, input_dim=43, kernel_initializer='normal', activation='relu'))
singlemodel.add(Dense(100, input_dim=43, kernel_initializer='normal', activation='relu'))
```

In [33]:
```
# Das Output Layer hat ein Neuronen
singlemodel.add(Dense(1, kernel_initializer='normal',activation='sigmoid'))
```

In [34]:
```
# Optimierungs- und Lossfunktion festlegen
singlemodel.compile(optimizer='rmsprop', loss='binary_crossentropy',metrics=['accuracy'])
```

In [35]:
```
# Training des Modells
singlemodel.fit(X_trainsingle, y_trainsingle, epochs=10, batch_size=25)
```

```
Epoch 1/10
702/702 [==============================] - 1s 763us/step - loss: nan - accuracy: 0.0112
Epoch 2/10
702/702 [==============================] - 1s 842us/step - loss: nan - accuracy: 0.0112
Epoch 3/10
702/702 [==============================] - 1s 842us/step - loss: nan - accuracy: 0.0112
Epoch 4/10
702/702 [==============================] - 1s 786us/step - loss: nan - accuracy: 0.0112
Epoch 5/10
702/702 [==============================] - 1s 970us/step - loss: nan - accuracy: 0.0112
Epoch 6/10
702/702 [==============================] - 1s 866us/step - loss: nan - accuracy: 0.0112
Epoch 7/10
702/702 [==============================] - 1s 1ms/step - loss: nan - accuracy: 0.0112
Epoch 8/10
702/702 [==============================] - 1s 1ms/step - loss: nan - accuracy: 0.0112
Epoch 9/10
702/702 [==============================] - 1s 991us/step - loss: nan - accuracy: 0.0112
Epoch 10/10
702/702 [==============================] - 1s 1ms/step - loss: nan - accuracy: 0.0112
```

Out[35]:
```
<tensorflow.python.keras.callbacks.History at 0x23406c80c88>
```

In [30]:
```
# Training des Modells
singlemodel.fit(X_trainsingle, y_trainsingle, epochs=50, batch_size=25)
```

```
Epoch 1/50
702/702 [==============================] - 1s 765us/step - loss: nan - accuracy: 0.0112
Epoch 2/50
702/702 [==============================] - 1s 729us/step - loss: nan - accuracy: 0.0112
Epoch 3/50
702/702 [==============================] - 0s 684us/step - loss: nan - accuracy: 0.0112
Epoch 4/50
702/702 [==============================] - 0s 684us/step - loss: nan - accuracy: 0.0112
Epoch 5/50
702/702 [==============================] - 0s 684us/step - loss: nan - accuracy: 0.0112
Epoch 6/50
702/702 [==============================] - 0s 695us/step - loss: nan - accuracy: 0.0112
Epoch 7/50
702/702 [==============================] - 0s 638us/step - loss: nan - accuracy: 0.0112
Epoch 8/50
702/702 [==============================] - 1s 718us/step - loss: nan - accuracy: 0.0112
Epoch 9/50
702/702 [==============================] - 0s 674us/step - loss: nan - accuracy: 0.0112
Epoch 10/50
702/702 [==============================] - 1s 721us/step - loss: nan - accuracy: 0.0112
Epoch 11/50
702/702 [==============================] - 0s 684us/step - loss: nan - accuracy: 0.0112
Epoch 12/50
702/702 [==============================] - 0s 699us/step - loss: nan - accuracy: 0.0112
Epoch 13/50
702/702 [==============================] - 0s 683us/step - loss: nan - accuracy: 0.0112
Epoch 14/50
702/702 [==============================] - 0s 702us/step - loss: nan - accuracy: 0.0112
Epoch 15/50
702/702 [==============================] - 0s 661us/step - loss: nan - accuracy: 0.0112
Epoch 16/50
702/702 [==============================] - 0s 707us/step - loss: nan - accuracy: 0.0112
Epoch 17/50
702/702 [==============================] - 0s 675us/step - loss: nan - accuracy: 0.0112
Epoch 18/50
702/702 [==============================] - 1s 722us/step - loss: nan - accuracy: 0.0112
Epoch 19/50
702/702 [==============================] - 0s 679us/step - loss: nan - accuracy: 0.0112
Epoch 20/50
702/702 [==============================] - 1s 715us/step - loss: nan - accuracy: 0.0112
Epoch 21/50
702/702 [==============================] - 0s 667us/step - loss: nan - accuracy: 0.0112
Epoch 22/50
702/702 [==============================] - 0s 700us/step - loss: nan - accuracy: 0.0112
Epoch 23/50
702/702 [==============================] - 1s 722us/step - loss: nan - accuracy: 0.0112
Epoch 24/50
702/702 [==============================] - 0s 707us/step - loss: nan - accuracy: 0.0112
Epoch 25/50
702/702 [==============================] - 0s 707us/step - loss: nan - accuracy: 0.0112
Epoch 26/50
702/702 [==============================] - 0s 707us/step - loss: nan - accuracy: 0.0112
Epoch 27/50
702/702 [==============================] - 0s 670us/step - loss: nan - accuracy: 0.0112
Epoch 28/50
702/702 [==============================] - 0s 686us/step - loss: nan - accuracy: 0.0112
Epoch 29/50
702/702 [==============================] - 0s 692us/step - loss: nan - accuracy: 0.0112
Epoch 30/50
702/702 [==============================] - 1s 720us/step - loss: nan - accuracy: 0.0112
Epoch 31/50
702/702 [==============================] - 0s 697us/step - loss: nan - accuracy: 0.0112
Epoch 32/50
702/702 [==============================] - 1s 718us/step - loss: nan - accuracy: 0.0112
Epoch 33/50
702/702 [==============================] - 0s 695us/step - loss: nan - accuracy: 0.0112
Epoch 34/50
702/702 [==============================] - 0s 695us/step - loss: nan - accuracy: 0.0112
Epoch 35/50
702/702 [==============================] - 0s 686us/step - loss: nan - accuracy: 0.0112
Epoch 36/50
702/702 [==============================] - 1s 1ms/step - loss: nan - accuracy: 0.0112
Epoch 37/50
702/702 [==============================] - 1s 1ms/step - loss: nan - accuracy: 0.0112
Epoch 38/50
702/702 [==============================] - 1s 774us/step - loss: nan - accuracy: 0.0112
Epoch 39/50
702/702 [==============================] - 1s 735us/step - loss: nan - accuracy: 0.0112
Epoch 40/50
702/702 [==============================] - 1s 753us/step - loss: nan - accuracy: 0.0112
Epoch 41/50
702/702 [==============================] - 1s 989us/step - loss: nan - accuracy: 0.0112
Epoch 42/50
702/702 [==============================] - 1s 752us/step - loss: nan - accuracy: 0.0112
Epoch 43/50
702/702 [==============================] - 1s 778us/step - loss: nan - accuracy: 0.0112
Epoch 44/50
```

```
702/702 [==============================] - 1s 782us/step - loss: nan - accuracy: 0.0112
Epoch 45/50
702/702 [==============================] - 1s 786us/step - loss: nan - accuracy: 0.0112
Epoch 46/50
702/702 [==============================] - 1s 800us/step - loss: nan - accuracy: 0.0112
Epoch 47/50
702/702 [==============================] - 1s 752us/step - loss: nan - accuracy: 0.0112
Epoch 48/50
702/702 [==============================] - 1s 765us/step - loss: nan - accuracy: 0.0112
Epoch 49/50
702/702 [==============================] - 1s 775us/step - loss: nan - accuracy: 0.0112
Epoch 50/50
702/702 [==============================] - 1s 796us/step - loss: nan - accuracy: 0.0112
```

Out[30]: `<tensorflow.python.keras.callbacks.History at 0x23406c3e188>`

In [36]:
```
# Vorbereitung der Testdaten
y_testsingle = df_test[['count_out']].values
```

In [37]:
```
X_testsingle = df_test[colssingle].values
```

In [38]:
```
# Werte in Floats umwandeln
y_testsingle = np.asarray(y_testsingle).astype(np.float32)
```

In [39]:
```
X_testsingle = np.asarray(X_testsingle).astype(np.float32)
```

In [41]:
```
# Evaluation des Modells mit den Test
scoresingle = singlemodel.evaluate(X_testsingle, y_testsingle)
print("Accuracy Score: "+str(round(scoresingle[1],4)))
```

```
272/272 [==============================] - 0s 622us/step - loss: nan - accuracy: 0.0025
Accuracy Score: 0.0025
```

Ergebnisse:

In der aktuellen Konfiguration hat das neuronale Netz mit mehreren Rückgabewerten bessere Accurancy Werte als das trainierte Neuronale Netz mit nur einem Rückgabewert.

Ausblick:

Das aktuelle Modell des neuronalen Netzes ist eher minimalistisch gestaltet um die Vorhersagegenauigkeit zu erhöhen, können folgende Parameter erhöht werden:

Die Anzahl der Zwischenschichten kann erhöht werden.

Die Anzahl der Neuronen je Zwischenschicht kann erhöht werden.

Unterschiedliche Optimierungsfunktionen und Loosfunktionen können getestet werden.

Die Aktivierungsfunktion der Neuronen im Neuronalen Netz kann ausgetauscht werden.

Folgende Werte können nicht ausgetauscht werden, da diese den Business Case ändern würden:

Die Anzahl der Input Neuronen

Die Anzahl der Output Neuronen

Da das Modell für den Business Case gut genug ist, werden diese Parameter nicht weiter getestet.

In [ ]: