Booklet_2_Code

August 8, 2020

[ ]:

```
import numpy as np
import pandas as pd
import time

import tensorflow as tf
from tensorflow import keras

import category_encoders as ce

from sklearn.model_selection import RandomizedSearchCV
```

1 Booklet 2 Neuronale Netze Aufgabe 2

1.0.1 Using RandomSearch to find the perfect Hyperparameter combination

[ ]:

```
# Erstelle Modell innerhalb einer Funktion, um es für Grid Search nutzen zu␣
 ↪können
def build_model(n_hidden = 1, n_neurons=30, learning_rate=0.1,␣
 ↪activation_function='relu', dropout_prop=0.25):
    model = keras.models.Sequential()
    model.add(keras.layers.Dense(2, activation='relu'))
    for layer in range(n_hidden):
        model.add(keras.layers.Dense(n_neurons, activation=activation_function))
    model.add(keras.layers.Dense(1))
    model.compile(optimizer=tf.keras.optimizers.
 ↪SGD(learning_rate=learning_rate),
                  loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                  metrics=['accuracy'])
    model.add(keras.layers.Dropout(dropout_prop))
    return model

nn = keras.wrappers.scikit_learn.KerasClassifier(build_model)
```

[ ]:

```
# zu untersuchende Parameter mit jeweiligem Parameterraum
params = {
    'n_hidden': [0,1,2,3,4],
    'n_neurons': [1,3,5,10,20,50,100],
    'learning_rate': [0.1, 0.05, 0.01],
    'activation_function': ['relu', 'sigmoid', 'elu'],
    'dropout_prop': [0, 0.25, 0.5]
}
```

[ ]:

```
random_search = RandomizedSearchCV(nn, params, n_iter=20)
random_search.fit(X_train.values,
                  y_train.values,
                  validation_data=(X_test.values, y_test.values),
                  callbacks=[keras.callbacks.EarlyStopping(patience=6)],
                  batch_size=32,
                  epochs=100
                 )
```

[ ]:

```
random_search.best_params_
```

1.1 Building the winner Model

[ ]:

```
#dont need this anymore...
train_dataset = tf.data.Dataset.from_tensor_slices((X_train.values, y_train.
 ↪values))
train_dataset = train_dataset.shuffle(len(X_train)).batch(32)

test_dataset = tf.data.Dataset.from_tensor_slices((X_test.values, y_test.
 ↪values))
test_dataset = test_dataset.shuffle(len(X_test)).batch(len(X_test))
```

[ ]:

```
nn = keras.models.Sequential()
nn.add(keras.layers.Dense(2, activation='elu'))
for layer in range(4):
        nn.add(keras.layers.Dense(3, activation='elu'))
nn.add(keras.layers.Dense(1))

nn.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.1),
           loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
           metrics=['accuracy'])
```

[ ]:

```
nn.fit(train_dataset,
       validation_steps=1,
       validation_data=test_dataset,
       batch_size=32,
       callbacks=[keras.callbacks.EarlyStopping(patience=6)],
       epochs=50)
```