Sanaz Salari

NON-SEPARABLE CLASSIFICATION ASSIGNMENT – MULTI-FONT CHARACTER RECOGNITION:

This problem is a multi-input and multi-output classification issue that we would like to solve by MLP algorithm. This is a classification problem which involves 26 classes as outputs and 6 inputs. The task here is to design and train an MLP to minimize the error between actual and predicted output values. We have 26*3 = 78 data points as training dataset and 26*3 = 78 datapoints as testing dataset. The steps I did from beginning to end will be explained in the following.

1. **install and import the required libraries** such as TensorFlow and Keras and fix random seed for reproducibility:

```
import sklearn
import skopt
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import RepeatedKFold
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.models import Sequential
from keras.layers import Dense, Dropout
np.random.seed(5)
```

2. **Import the data** from the local drive and do preprocess. Split train and test datasets as two .csv files on a local drive and import them as follows:

```
import pandas as pd
import io
train0 = pd.read_csv(io.BytesIO(uploaded['font_train.csv']))
test0 = pd.read_csv(io.BytesIO(uploaded['font_test.csv']))
train0.head()
train0.shape
test0.shape
```

	input1	input2	input3	input4	input5	input6	input7	input8	input9	input10	...	Q	R	S	T	U	V	W	X	Y	Z
0	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	12	10	1	1	0	0	0	4	6	0	...	0	0	0	0	0	0	0	0	0	0
3	21	10	4	4	0	1	1	0	5	0	...	0	0	0	0	0	0	0	0	0	0
4	27	12	3	3	0	8	0	5	0	2	...	0	0	0	0	0	0	0	0	0	0

5 rows × 40 columns

3. **Check the types of input and output values**

   Types of data in both test and train datasets are checked to see whether we have any categorical data and if there are such values, convert them to numerical ones. All data types are float and integer.

```
train0.dtypes
test0.dtypes
```

4. **Check any NA values**

   We don’t have any NA values in this dataset.

```
train0.isna().sum()
test0.isna().sum()
```

5. **Find duplicated values in the features if we have**

   In the training dataset, rows 13 and 14 are duplicated.

```
df1 = train0.duplicated()
duplicate = train0.loc[df1 == True]
print(duplicate)
```

   In the test dataset, rows 73 and 74 are duplicated.

```
df = test0.duplicated()
duplicate = test0.loc[df == True]
print(duplicate)
```

```
    input1  input2  input3  input4  input5  input6  input7  input8  input9  \
73       0       6       6       6       0       0       0       0       0

    input10  ...  Q  R  S  T  U  V  W  X  Y  Z
73        0  ...  0  0  0  0  0  0  0  0  1  0

[1 rows x 40 columns]
```

6. **Drop Duplicated rows from training**
   (rows 13 and 14 are duplicated and row 14 is removed)

```
train1 = train0.drop([train0.index[14]])
train1[train1.duplicated(['input1','input2','input3','input4','input
5','input6','input7','input8','input9','input10','input11','input12'
,'input13','input14'])]
print(train1.duplicated().sum())
```

7. **#Drop Duplicated rows from testing(rows 73 and 74 are duplicated and r ow 73 is removed)**

```
test1 = test0.drop([test0.index[73]])
test1[test1.duplicated(['input1','input2','input3','input4','input5'
,'input6','input7','input8','input9','input10','input11','input12','
input13','input14'])]
print(test1.duplicated().sum())
test1.tail()
```

8. **Inspect the data to see whether it needs to do normalization or not**

```
train1_stats = train1.describe()
train1_stats = train1_stats.transpose()
train1_stats.head()
```

	count	mean	std	min	25%	50%	75%	max
input1	77.0	7.870130	7.347306	0.0	0.0	8.0	12.0	27.0
input2	77.0	11.389610	6.947464	0.0	9.0	11.0	15.0	31.0
input3	77.0	2.337662	2.552658	0.0	0.0	2.0	4.0	10.0
input4	77.0	2.558442	2.935686	0.0	0.0	2.0	4.0	12.0
input5	77.0	0.194805	0.973855	0.0	0.0	0.0	0.0	5.0

9. **Normalization**

To decrease the training model sensitivity to features scale, we need to normalize all training features (not outputs) which have different ranges to change their values to a common scale. Normalization is important because the input variables will be multiplied by the model weights and the scale of outputs will be affected by the scale of features. The goal is to predict output with the best accuracy, but we don't want to change the output.

Training data normalization:

```
#Normalization
train2=(train0 - train0.min())/(train0.max() - train0.min())
train2.head()
```

	input1	input2	input3	input4	input5	input6	input7	input8	input9	input10	...	Q	R	S	T	U	V	W	X	Y	Z
0	0.296296	0.000000	0.0	0.000000	0.0	0.000000	0.000	0.0	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.259259	0.000000	0.0	0.000000	0.0	0.000000	0.000	0.0	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.444444	0.322581	0.1	0.083333	0.0	0.000000	0.000	0.8	0.666667	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.777778	0.322581	0.4	0.333333	0.0	0.090909	0.125	0.0	0.555556	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	1.000000	0.387097	0.3	0.250000	0.0	0.727273	0.000	1.0	0.000000	0.666667	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 40 columns

Test data normalization:

```
#Normalization
test2=(test0 - train0.min())/(train0.max() - train0.min())
test2.head()
```

	input1	input2	input3	input4	input5	input6	input7	input8	input9	input10	...	Q	R	S	T	U	V	W	X	Y	Z
0	0.259259	0.000000	0.0	0.000000	0.0	0.000000	0.000	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.259259	0.000000	0.0	0.000000	0.0	0.000000	0.000	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.222222	0.161290	0.0	0.500000	0.0	0.000000	1.125	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1.000000	0.451613	0.2	0.166667	0.0	0.727273	0.000	1.0	0.0	0.666667	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	1.000000	0.387097	0.2	0.166667	0.0	0.727273	0.000	1.0	0.0	0.666667	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 40 columns

10. Split features from labels for training and testing datasets

Separate the target values from the features. These targets are the values that we will train the model to predict.

```
[25] #Split features from labels
     #Separate the target values from the features. This targets are the values that we will train the model to predict.
     train2_features = train2.copy()
     test2_features = test2.copy()
     train2_labels = train2_features.iloc[:, 14:40]

     train2_features.drop(['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'], axis = 1, inplace = True)
     print(train2_features.shape)
     print(train2_labels.shape)
```

```
(78, 14)
(78, 26)
```

```
#Split features from labels
#Separate the target values from the features. This targets are the values that we will train the model to predict.
test2_features = test2.copy()
test2_labels = test2_features.iloc[:, 14:40]

test2_features.drop(['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'], axis = 1, inplace = True)
print(test2_features.shape)
print(test2_labels.shape)
```

```
(78, 14)
(78, 26)
```

11.Change train and test datasets from matrix to array:

```
# Change train and test datasets from matrix to array:
X_train = train2_features.to_numpy()
y_train = train2_labels.to_numpy()
X_test = test2_features.to_numpy()
y_test = test2_labels.to_numpy()
X_train = X_train.reshape([X_train.shape[0], -1])
X_test = X_test.reshape([X_test.shape[0], -1])

y_train= train2_labels
y_test= test2_labels
# Printing dimensions
print(X_train.shape, y_train.shape)
```

```
(77, 14) (77, 26)
```

12. Define a model with a single layer

The model is defined with sequential keras syntax. As we told earlier, we have 26 outputs and 14 features. Since the problem is classification, the activation function is chosen as “sigmoid” and for the compile part, the optimizer is “adam”, the loss function is “categorical crossentropy” and the metrics is “accuracy”.

```
#Define a model with a single layer
model = keras.Sequential([
                    keras.layers.Dense(26, input_shape=(14,), activation='sigmoid')
])

model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model.fit(X_train, y_train, epochs=50)
```

The result of running the model will be as follows:

```
3/3 [==============================] - 0s 5ms/step - loss: -2.6211 - accuracy: 0.1026
Epoch 38/50
3/3 [==============================] - 0s 3ms/step - loss: -2.7200 - accuracy: 0.1026
Epoch 39/50
3/3 [==============================] - 0s 4ms/step - loss: -2.8183 - accuracy: 0.1026
Epoch 40/50
3/3 [==============================] - 0s 4ms/step - loss: -2.9168 - accuracy: 0.1282
Epoch 41/50
3/3 [==============================] - 0s 5ms/step - loss: -3.0138 - accuracy: 0.1282
Epoch 42/50
3/3 [==============================] - 0s 4ms/step - loss: -3.1122 - accuracy: 0.1282
Epoch 43/50
3/3 [==============================] - 0s 5ms/step - loss: -3.2120 - accuracy: 0.1282
Epoch 44/50
3/3 [==============================] - 0s 4ms/step - loss: -3.3079 - accuracy: 0.1282
Epoch 45/50
3/3 [==============================] - 0s 4ms/step - loss: -3.4067 - accuracy: 0.1410
Epoch 46/50
3/3 [==============================] - 0s 4ms/step - loss: -3.5031 - accuracy: 0.1410
Epoch 47/50
3/3 [==============================] - 0s 4ms/step - loss: -3.6010 - accuracy: 0.1538
Epoch 48/50
3/3 [==============================] - 0s 4ms/step - loss: -3.6987 - accuracy: 0.1538
Epoch 49/50
3/3 [==============================] - 0s 6ms/step - loss: -3.7961 - accuracy: 0.1667
Epoch 50/50
3/3 [==============================] - 0s 6ms/step - loss: -3.8935 - accuracy: 0.1667
<keras.callbacks.History at 0x7f757e3a2750>
```

As we see in the result window, loss value is about -3.8 and accuracy is about 0.16. The loss value is too large, and the accuracy is too low. So, we need to improve the model.

13. Evaluate the model on the test dataset

```
#Evaluate model on the test dataset
model.evaluate(X_test, y_test)
```

```
3/3 [==============================] - 0s 7ms/step - loss: -3.5059 - accuracy: 0.1282
[-3.5058586597442627, 0.12820513546466827]
```

14. sample prediction

```
#sample prediction
y_predicted = model.predict(X_test)
y_predicted[2]
```

```
array([0.07601583, 0.4537092 , 0.43642277, 0.29597253, 0.15798795,
       0.07947573, 0.67117137, 0.43142015, 0.13583308, 0.8494957 ,
       0.6678968 , 0.832185  , 0.60348904, 0.11200804, 0.22369573,
       0.06757528, 0.5559669 , 0.88929534, 0.29677683, 0.6702946 ,
       0.9051914 , 0.89692247, 0.30873224, 0.16228747, 0.5716726 ,
       0.9486872 ], dtype=float32)
```

As we see, since the model accuracy is too low, it cannot predict the target value accurately.

15. To improve the model, we will add hidden layers

Let's create a helper function that builds the model with various parameters. Builds a Sequential MLP model using Keras.

The model has two hidden layers and one output layer.

Activation function for each hidden layer is “Relu” and for the output layer is “sigmoid” since we have a classification problem.

As we want to solve a regularization problem, the loss function is “cross entropy”.

Regularization method is “He Uniform” for hidden layers as we use “Relu” function and “GlorotUniform” for the output layer as we use “sigmoid” function in the output layer.

```
#To improve model, we will add hidden layers
from tensorflow.keras import regularizers
from tensorflow.keras import layers
from tensorflow.keras import initializers
model = keras.Sequential([
                    keras.layers.Dense(20, input_shape=(14,), activation='relu',
                    kernel_regularizer=regularizers.l1_l2(l1=1e-4, l2=1e-4),
                    bias_regularizer=regularizers.l2(1e-4),
                    activity_regularizer=regularizers.l2(1e-4),
                    kernel_initializer = tf.keras.initializers.HeUniform(),
                    name="dense_1"),
                    keras.layers.Dense(20, input_shape=(14,), activation='relu',
                    kernel_regularizer=regularizers.l1_l2(l1=1e-4, l2=1e-4),
                    bias_regularizer=regularizers.l2(1e-4),
                    activity_regularizer=regularizers.l2(1e-4),
                    kernel_initializer = tf.keras.initializers.HeUniform(),
                    name="dense_2"),
                    keras.layers.Dense(26, activation='sigmoid',kernel_initializer = tf.keras.initializers.GlorotUniform())
])

model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model.fit(X_train, y_train, epochs=200)
```

The result would be as follows:

```
3/3 [==============================] - 0s 4ms/step - loss: 0.6291 - accuracy: 0.9103
Epoch 185/200
3/3 [==============================] - 0s 4ms/step - loss: 0.6234 - accuracy: 0.9231
Epoch 186/200
3/3 [==============================] - 0s 4ms/step - loss: 0.6174 - accuracy: 0.9103
Epoch 187/200
3/3 [==============================] - 0s 4ms/step - loss: 0.6123 - accuracy: 0.9103
Epoch 188/200
3/3 [==============================] - 0s 4ms/step - loss: 0.6062 - accuracy: 0.9103
Epoch 189/200
3/3 [==============================] - 0s 4ms/step - loss: 0.6005 - accuracy: 0.9103
Epoch 190/200
3/3 [==============================] - 0s 4ms/step - loss: 0.5955 - accuracy: 0.9103
Epoch 191/200
3/3 [==============================] - 0s 4ms/step - loss: 0.5900 - accuracy: 0.9231
Epoch 192/200
3/3 [==============================] - 0s 4ms/step - loss: 0.5850 - accuracy: 0.9231
Epoch 193/200
3/3 [==============================] - 0s 4ms/step - loss: 0.5803 - accuracy: 0.9231
Epoch 194/200
3/3 [==============================] - 0s 6ms/step - loss: 0.5752 - accuracy: 0.9231
Epoch 195/200
3/3 [==============================] - 0s 4ms/step - loss: 0.5703 - accuracy: 0.9231
Epoch 196/200
3/3 [==============================] - 0s 3ms/step - loss: 0.5657 - accuracy: 0.9231
Epoch 197/200
3/3 [==============================] - 0s 3ms/step - loss: 0.5611 - accuracy: 0.9231
Epoch 198/200
3/3 [==============================] - 0s 4ms/step - loss: 0.5560 - accuracy: 0.9231
Epoch 199/200
3/3 [==============================] - 0s 4ms/step - loss: 0.5522 - accuracy: 0.9231
Epoch 200/200
3/3 [==============================] - 0s 3ms/step - loss: 0.5470 - accuracy: 0.9231
<keras.callbacks.History at 0x7f756d673fd0>
```

Loss value decreases and accuracy increases by each epoch which shows no overfitting or underfitting.

16. Evaluate model on the test dataset

Although test prediction shows a higher loss and lower accuracy for the testing dataset, there is no overfitting or underfitting since the loss and accuracy between training and test results are small.

```
#Evaluate model on the test dataset
model.evaluate(X_test, y_test)
```

```
3/3 [==============================] - 0s 4ms/step - loss: 1.1658 - accuracy: 0.7821
[1.1658493280410767, 0.7820512652397156]
```