AI Assignment 2

Name : Gangisetty Krishna Vamsi

Reg.No : 20BDS0281

college : VIT Vellore

email :gangikrishna.vamsi2020@vitstudent.ac.in

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split


df = pd.read_csv('drug200.csv')
```

Task 1 : Read the dataset and do data pre-processing

```
label_encoder = LabelEncoder()
df['Sex'] = label_encoder.fit_transform(df['Sex'])
df['BP'] = label_encoder.fit_transform(df['BP'])
df['Cholesterol'] = label_encoder.fit_transform(df['Cholesterol'])
df['Drug'] = label_encoder.fit_transform(df['Drug'])
print(df.head())
```

```
   Age  Sex  BP  Cholesterol  Na_to_K  Drug
0   23    0   0            0   25.355     0
1   47    1   1            0   13.093     3
2   47    1   1            0   10.114     3
3   28    0   2            0    7.798     4
4   61    0   1            0   18.043     0
```

```
# Scale numerical variables
scaler = StandardScaler()
df[['Age', 'Na_to_K']] = scaler.fit_transform(df[['Age', 'Na_to_K']])


# Separate features and labels
X = df[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']]
y = df['Drug']


# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)


print(X_train.shape)
print(y_test.shape)
```

```
(160, 5)
(40,)
```

Task 2 : Build the ANN model with (input layer, min 3 hidden layers & output layer)

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


# Define the model architecture
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(5,)))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(5, activation='softmax'))


x = df.iloc[:,0:5]
y = df.iloc[:,5:]
print(x)
print(y)
```

```
          Age  Sex  BP  Cholesterol   Na_to_K
0   -1.291591    0   0            0  1.286522
```

```
1    0.162699    1   1            0 -0.415145
2    0.162699    1   1            0 -0.828558
3   -0.988614    0   2            0 -1.149963
4    1.011034    0   1            0  0.271794
..        ...  ...  ..          ...       ...
195  0.708057    0   1            0 -0.626917
196 -1.715759    1   1            0 -0.565995
197  0.465676    1   2            0 -0.859089
198 -1.291591    1   2            1 -0.286500
199 -0.261469    0   1            1 -0.657170

[200 rows x 5 columns]
     Drug
0       0
1       3
2       3
3       4
4       0
..    ...
195     3
196     3
197     4
198     4
199     4

[200 rows x 1 columns]
```

```
# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',metrics=['accuracy'])
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
model.fit(X_train, y_train_encoded, epochs=20, batch_size=20,validation_data=(X_test, y_test_encoded))
```

```
Epoch 1/20
8/8 [==============================] - 1s 30ms/step - loss: 1.5333 - accuracy: 0.4375 - val_loss: 1.4609 - val_accuracy: 0.5750
Epoch 2/20
8/8 [==============================] - 0s 7ms/step - loss: 1.3591 - accuracy: 0.6687 - val_loss: 1.3021 - val_accuracy: 0.6000
Epoch 3/20
8/8 [==============================] - 0s 4ms/step - loss: 1.1273 - accuracy: 0.6750 - val_loss: 1.0769 - val_accuracy: 0.6000
Epoch 4/20
8/8 [==============================] - 0s 4ms/step - loss: 0.8779 - accuracy: 0.6875 - val_loss: 0.8743 - val_accuracy: 0.6250
Epoch 5/20
8/8 [==============================] - 0s 4ms/step - loss: 0.6731 - accuracy: 0.7625 - val_loss: 0.7003 - val_accuracy: 0.8000
Epoch 6/20
8/8 [==============================] - 0s 4ms/step - loss: 0.5185 - accuracy: 0.8875 - val_loss: 0.5997 - val_accuracy: 0.8250
Epoch 7/20
8/8 [==============================] - 0s 4ms/step - loss: 0.4107 - accuracy: 0.9000 - val_loss: 0.5079 - val_accuracy: 0.8500
Epoch 8/20
8/8 [==============================] - 0s 5ms/step - loss: 0.3332 - accuracy: 0.9125 - val_loss: 0.4127 - val_accuracy: 0.8750
Epoch 9/20
8/8 [==============================] - 0s 5ms/step - loss: 0.2748 - accuracy: 0.9312 - val_loss: 0.3762 - val_accuracy: 0.8750
Epoch 10/20
8/8 [==============================] - 0s 4ms/step - loss: 0.2377 - accuracy: 0.9062 - val_loss: 0.3104 - val_accuracy: 0.8750
Epoch 11/20
8/8 [==============================] - 0s 4ms/step - loss: 0.2018 - accuracy: 0.9438 - val_loss: 0.2988 - val_accuracy: 0.9000
Epoch 12/20
8/8 [==============================] - 0s 5ms/step - loss: 0.1778 - accuracy: 0.9500 - val_loss: 0.2459 - val_accuracy: 0.9000
Epoch 13/20
8/8 [==============================] - 0s 7ms/step - loss: 0.1556 - accuracy: 0.9438 - val_loss: 0.2310 - val_accuracy: 0.9000
Epoch 14/20
8/8 [==============================] - 0s 4ms/step - loss: 0.1310 - accuracy: 0.9625 - val_loss: 0.2035 - val_accuracy: 0.9250
Epoch 15/20
8/8 [==============================] - 0s 4ms/step - loss: 0.1194 - accuracy: 0.9688 - val_loss: 0.1884 - val_accuracy: 0.9000
Epoch 16/20
8/8 [==============================] - 0s 4ms/step - loss: 0.1062 - accuracy: 0.9688 - val_loss: 0.1681 - val_accuracy: 0.9250
Epoch 17/20
8/8 [==============================] - 0s 4ms/step - loss: 0.0953 - accuracy: 0.9812 - val_loss: 0.1527 - val_accuracy: 0.9500
Epoch 18/20
8/8 [==============================] - 0s 4ms/step - loss: 0.0890 - accuracy: 0.9688 - val_loss: 0.1443 - val_accuracy: 0.9750
Epoch 19/20
8/8 [==============================] - 0s 7ms/step - loss: 0.0767 - accuracy: 0.9812 - val_loss: 0.1205 - val_accuracy: 0.9750
Epoch 20/20
8/8 [==============================] - 0s 5ms/step - loss: 0.0720 - accuracy: 0.9750 - val_loss: 0.1176 - val_accuracy: 0.9750
<keras.callbacks.History at 0x7f78df139960>
```

```
y_pred = model.predict(X_test)
y_pred
```

```
 6.0714574e-05],
[2.9487553e-04, 8.7926345e-04, 9.8734874e-01, 1.0748758e-02,
 7.2838779e-04],
[2.7253757e-08, 4.0693404e-09, 2.2954080e-08, 2.3762800e-03,
 9.9762356e-01],
[2.3553293e-04, 2.5890194e-02, 5.0770788e-04, 5.6319612e-01,
 4.1017041e-01],
[9.9999994e-01, 1.9278785e-14, 2.8700911e-12, 1.0069469e-09,
 8.5004265e-10],
[7.6509267e-03, 2.1926614e-02, 9.0725315e-01, 4.2679988e-02,
 2.0489221e-02],
[8.7648799e-04, 6.6970133e-06, 4.0172562e-03, 4.9075127e-02,
 9.4602454e-01],
[2.8480617e-05, 2.1350401e-04, 9.8549752e-03, 3.6977911e-01,
 6.2012398e-01],
[9.9999994e-01, 6.8119029e-13, 2.4704632e-11, 2.1036108e-09,
 3.0876596e-10],
[9.9999994e-01, 5.2891437e-16, 2.0848967e-13, 5.3569989e-11,
 7.8245958e-11],
[9.9999994e-01, 3.8596754e-13, 1.3642829e-11, 1.3714172e-09,
 1.7119328e-10],
[1.3968181e-04, 2.4151718e-03, 2.6768185e-03, 6.4586771e-01,
 3.4890047e-01],
[1.5817126e-04, 5.3710619e-07, 1.8748597e-08, 1.4507201e-02,
 9.8533410e-01],
[9.9999934e-01, 2.7884144e-09, 1.6211013e-08, 5.1237322e-07,
 6.2431496e-08],
[4.1437000e-02, 7.9180045e-06, 6.6709894e-05, 1.6789062e-02,
 9.4169933e-01],
[9.9998719e-01, 3.2134368e-09, 5.0280512e-08, 4.1720559e-06,
 8.5626098e-06],
[2.0791229e-05, 2.4659359e-01, 6.9638743e-05, 6.9118565e-01,
 6.2130313e-02],
[2.9126883e-01, 7.9072146e-03, 8.9886400e-04, 3.1544948e-01,
 3.8447568e-01],
[9.9999839e-01, 7.4578754e-10, 7.2663147e-09, 8.0164205e-07,
 7.5428125e-07],
[2.5054128e-04, 8.2321990e-01, 1.2009491e-01, 5.5251271e-02,
 1.1833678e-03],
[9.9999994e-01, 1.2828186e-11, 1.4888346e-10, 1.1260811e-08,
 1.0123351e-09],
[5.3313216e-03, 4.2984243e-06, 7.6709990e-03, 5.3216867e-02,
 9.3377650e-01],
[2.9460595e-03, 9.6945000e-01, 7.1064290e-03, 1.8467085e-02,
 2.0303708e-03],
[9.9997967e-01, 1.0757916e-06, 3.9803149e-07, 1.7335757e-05,
 1.6034231e-06],
[7.5534347e-04, 9.8383874e-01, 4.0202141e-03, 1.0688832e-02,
 6.9687009e-04]], dtype=float32)
```

```
comp = pd.DataFrame(y_test_encoded) # Creating a dataframe
comp.columns = ['Actual Value'] # Changing the column name
comp
```

	Actual Value
0	4
1	0
2	4
3	3
4	0
5	0
6	0
7	4
8	1
9	4
10	1
11	4
12	0
13	1
14	2
15	0
16	2
17	4
18	3
19	0

```
# Print the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                384

 dense_1 (Dense)             (None, 128)               8320

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 32)                2080

 dense_4 (Dense)             (None, 5)                 165

=================================================================
Total params: 19,205
Trainable params: 19,205
Non-trainable params: 0
_________________________________________________________________
```

33	0

Task 3 : Test the model with random data

```
# Generate random data for testing
random_data = np.random.rand(1, 5)
random_data
```

```
array([[0.58857633, 0.45966327, 0.11338478, 0.10455691, 0.29322196]])
```

```
# Make predictions
predictions = model.predict(random_data)
predictions
```

```
1/1 [==============================] - 0s 49ms/step
array([[9.9934596e-01, 1.7040064e-05, 1.7047777e-04, 3.6303891e-04,
        1.0353992e-04]], dtype=float32)
```

```
# Get the predicted drug class
predicted_class = np.argmax(predictions)
```

```
# Print the predicted class
print("Predicted Drug Class :", predicted_class)
```

```
Predicted Drug Class : 0
```