

▼ Data Science Lab Assignment 8

Name:P Manohar Rao


Roll No: 197158

Section: CSE-A

- ▼ Implement a Simple Neural Network Model and predict if a person will buy insurance or not for the given dataset.

```
import pandas as pd
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

```
data = pd.read_csv("insurance_data.csv")
data.head()
```

	age	Affordability	bought_insurance	
0	22	1	0	
1	25	1	0	
2	47	0	1	
3	52	1	0	
4	46	1	1	

```
x = data.iloc[:, :-1]
y = data.iloc[:, -1]
print(x.shape)
```

```
(27, 2)
```

```
train_x, test_x, train_y, test_y = train_test_split(x, y, train_size=0.8, shuffle=True)
```

```
scaling = StandardScaler()
```

```
train_x = scaling.fit_transform(train_x)
test_x = scaling.fit_transform(test_x)
```

```

model = Sequential()
model.add(Dense(2,input_dim=2, activation='relu'))
model.add(Dense(2,input_dim=2, activation='relu'))
model.add(Dense(1,activation='softmax'))

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

model.fit(train_x, train_y, epochs=100, batch_size=40)
model.evaluate(test_x, test_y)
predict_y = model.predict(test_x)
actual_result = test_y.tolist()
predicted_result = predict_y.tolist()

1/1 [=====] - 0s 9ms/step - loss: 0.6914 - accuracy: 0.47 ▲
Epoch 45/100
1/1 [=====] - 0s 8ms/step - loss: 0.6907 - accuracy: 0.47
Epoch 46/100
1/1 [=====] - 0s 8ms/step - loss: 0.6900 - accuracy: 0.47
Epoch 47/100
1/1 [=====] - 0s 9ms/step - loss: 0.6893 - accuracy: 0.47
Epoch 48/100
1/1 [=====] - 0s 10ms/step - loss: 0.6886 - accuracy: 0.4
Epoch 49/100
1/1 [=====] - 0s 11ms/step - loss: 0.6879 - accuracy: 0.4
Epoch 50/100
1/1 [=====] - 0s 7ms/step - loss: 0.6873 - accuracy: 0.47
Epoch 51/100
1/1 [=====] - 0s 7ms/step - loss: 0.6866 - accuracy: 0.47
Epoch 52/100
1/1 [=====] - 0s 7ms/step - loss: 0.6859 - accuracy: 0.47
Epoch 53/100
1/1 [=====] - 0s 7ms/step - loss: 0.6852 - accuracy: 0.47
Epoch 54/100
1/1 [=====] - 0s 6ms/step - loss: 0.6846 - accuracy: 0.47
Epoch 55/100
1/1 [=====] - 0s 7ms/step - loss: 0.6839 - accuracy: 0.47
Epoch 56/100
1/1 [=====] - 0s 7ms/step - loss: 0.6833 - accuracy: 0.47
Epoch 57/100
1/1 [=====] - 0s 7ms/step - loss: 0.6826 - accuracy: 0.47
Epoch 58/100
1/1 [=====] - 0s 7ms/step - loss: 0.6820 - accuracy: 0.47
Epoch 59/100
1/1 [=====] - 0s 7ms/step - loss: 0.6813 - accuracy: 0.47
Epoch 60/100
1/1 [=====] - 0s 6ms/step - loss: 0.6807 - accuracy: 0.47
Epoch 61/100
1/1 [=====] - 0s 8ms/step - loss: 0.6801 - accuracy: 0.47
Epoch 62/100
1/1 [=====] - 0s 6ms/step - loss: 0.6795 - accuracy: 0.47
Epoch 63/100
1/1 [=====] - 0s 7ms/step - loss: 0.6788 - accuracy: 0.47
Epoch 64/100
1/1 [=====] - 0s 7ms/step - loss: 0.6782 - accuracy: 0.47
Epoch 65/100
1/1 [=====] - 0s 7ms/step - loss: 0.6776 - accuracy: 0.47
Epoch 66/100

```

```

Epoch 66/100
1/1 [=====] - 0s 7ms/step - loss: 0.6770 - accuracy: 0.47
Epoch 67/100
1/1 [=====] - 0s 7ms/step - loss: 0.6764 - accuracy: 0.47
Epoch 68/100
1/1 [=====] - 0s 15ms/step - loss: 0.6758 - accuracy: 0.47
Epoch 69/100
1/1 [=====] - 0s 9ms/step - loss: 0.6752 - accuracy: 0.47
Epoch 70/100
1/1 [=====] - 0s 10ms/step - loss: 0.6746 - accuracy: 0.47
Epoch 71/100
1/1 [=====] - 0s 10ms/step - loss: 0.6740 - accuracy: 0.47
Epoch 72/100
1/1 [=====] - 0s 17ms/step - loss: 0.6734 - accuracy: 0.47
Epoch 73/100

```

```
accuracy_score(predicted_result, actual_result)*100
```

```
66.66666666666666
```

- ▼ Use the given dataset and build a customer churn prediction model using artificial neural network.

```

import pandas as pd
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split as split_data
from sklearn.metrics import accuracy_score as accuracy
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

```

```

dataset = pd.read_csv("Churn_Modelling.csv")
dataset.head()

```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Ba
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	831
2	3	15619304	Onio	502	France	Female	42	8	1590
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	1251



```
dataset.drop(dataset.columns[[0,1,2]], axis=1, inplace=True)
```

```
dataset.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
0	619	France	Female	42	2	0.00	1	1
1	608	Spain	Female	41	1	83807.86	1	0
2	502	France	Female	42	8	159660.80	3	1
3	699	France	Female	39	1	0.00	2	0
4	850	Spain	Female	43	2	125510.82	1	1

```
def convert(vector):
    visited = []
    ind = 0
    result = []
    for i in vector:
        if i in visited:
            result.append(visited.index(i))
        else:
            visited.append(i)
            result.append(len(visited)-1)
    return result
```

```
dataset.iloc[:, 1] = np.array(convert(dataset.iloc[:, 1].tolist()))
dataset.iloc[:, 2] = np.array(convert(dataset.iloc[:, 2].tolist()))
```

```
dataset.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
0	619	0	0	42	2	0.00	1	1
1	608	1	0	41	1	83807.86	1	0
2	502	0	0	42	8	159660.80	3	1
3	699	0	0	39	1	0.00	2	0
4	850	1	0	43	2	125510.82	1	1

```
x = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
x.shape, y.shape

((10000, 10), (10000,))
```

```
train_x, test_x, train_y, test_y = split_data(x, y, train_size=0.75, shuffle=True)

scaling = StandardScaler()

train_x = scaling.fit_transform(train_x)
test_x = scaling.fit_transform(test_x)
```

```
model = Sequential()
model.add(Dense(10, input_dim=10, activation='relu', name='input'))
model.add(Dense(10, input_dim=10, activation='relu', name='layer1'))
model.add(Dense(5, input_dim=10, activation='relu', name='layer2'))
model.add(Dense(3, input_dim=5, activation='relu', name='layer3'))
model.add(Dense(1, activation='sigmoid', name='output'))
```

```
model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
input (Dense)	(None, 10)	110
layer1 (Dense)	(None, 10)	110
layer2 (Dense)	(None, 5)	55
layer3 (Dense)	(None, 3)	18
output (Dense)	(None, 1)	4

=====
Total params: 297
Trainable params: 297
Non-trainable params: 0
=====

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

```
def sigmoid_correction(predict):
    for i in range(len(predict)):
        predict[i] = 1 if predict[i][0] >=0.5 else 0
    return predict
```

```
model.fit(train_x, train_y, epochs=500, batch_size=450)
prediction = model.predict(test_x)
# model.evaluate(test_x, test_y)
actual_y = test_y.tolist()
predicted_y = sigmoid_correction(prediction.tolist())
```

```
Epoch 100/500
17/17 [=====] - 0s 2ms/step - loss: 0.3358 - accuracy: 0.
Epoch 106/500
17/17 [=====] - 0s 2ms/step - loss: 0.3361 - accuracy: 0.
Epoch 107/500
17/17 [=====] - 0s 2ms/step - loss: 0.3351 - accuracy: 0.
```

```

17/17 [=====] - 0s 2ms/step - loss: 0.3351 - accuracy: 0.
Epoch 108/500
17/17 [=====] - 0s 2ms/step - loss: 0.3351 - accuracy: 0.
Epoch 109/500
17/17 [=====] - 0s 2ms/step - loss: 0.3353 - accuracy: 0.
Epoch 110/500
17/17 [=====] - 0s 2ms/step - loss: 0.3351 - accuracy: 0.
Epoch 111/500
17/17 [=====] - 0s 2ms/step - loss: 0.3349 - accuracy: 0.
Epoch 112/500
17/17 [=====] - 0s 2ms/step - loss: 0.3345 - accuracy: 0.
Epoch 113/500
17/17 [=====] - 0s 2ms/step - loss: 0.3346 - accuracy: 0.
Epoch 114/500
17/17 [=====] - 0s 3ms/step - loss: 0.3345 - accuracy: 0.
Epoch 115/500
17/17 [=====] - 0s 2ms/step - loss: 0.3343 - accuracy: 0.
Epoch 116/500
17/17 [=====] - 0s 2ms/step - loss: 0.3344 - accuracy: 0.
Epoch 117/500
17/17 [=====] - 0s 2ms/step - loss: 0.3342 - accuracy: 0.
Epoch 118/500
17/17 [=====] - 0s 2ms/step - loss: 0.3340 - accuracy: 0.
Epoch 119/500
17/17 [=====] - 0s 3ms/step - loss: 0.3338 - accuracy: 0.
Epoch 120/500
17/17 [=====] - 0s 2ms/step - loss: 0.3337 - accuracy: 0.
Epoch 121/500
17/17 [=====] - 0s 3ms/step - loss: 0.3343 - accuracy: 0.
Epoch 122/500
17/17 [=====] - 0s 2ms/step - loss: 0.3339 - accuracy: 0.
Epoch 123/500
17/17 [=====] - 0s 2ms/step - loss: 0.3335 - accuracy: 0.
Epoch 124/500
17/17 [=====] - 0s 2ms/step - loss: 0.3333 - accuracy: 0.
Epoch 125/500
17/17 [=====] - 0s 2ms/step - loss: 0.3332 - accuracy: 0.
Epoch 126/500
17/17 [=====] - 0s 2ms/step - loss: 0.3329 - accuracy: 0.
Epoch 127/500
17/17 [=====] - 0s 2ms/step - loss: 0.3327 - accuracy: 0.
Epoch 128/500
17/17 [=====] - 0s 2ms/step - loss: 0.3330 - accuracy: 0.
Epoch 129/500
17/17 [=====] - 0s 2ms/step - loss: 0.3324 - accuracy: 0.
Epoch 130/500
17/17 [=====] - 0s 3ms/step - loss: 0.3324 - accuracy: 0.
Epoch 131/500
17/17 [=====] - 0s 2ms/step - loss: 0.3324 - accuracy: 0.
Epoch 132/500
17/17 [=====] - 0s 2ms/step - loss: 0.3328 - accuracy: 0.
Epoch 133/500
17/17 [=====] - 0s 2ms/step - loss: 0.3322 - accuracy: 0.

```

accuracy(predicted_y, actual_y)*100