IIS Project Title: Bank Customer Churn Prediction

Importing the libraries

```
import numpy as np
import pandas as pd
import tensorflow as tf
```

Part 1 - Data Preprocessing

Importing the dataset

```
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:-1].values
y = dataset.iloc[:, -1].values
```

```
print(X)
```

```
[[619 'France' 'Female' ... 1 1 101348.88]
 [608 'Spain' 'Female' ... 0 1 112542.58]
 [502 'France' 'Female' ... 1 0 113931.57]
 ...
 [709 'France' 'Female' ... 0 1 42085.58]
 [772 'Germany' 'Male' ... 1 0 92888.52]
 [792 'France' 'Female' ... 1 0 38190.78]]
```

```
print(y)
```

```
[1 0 1 ... 1 1 0]
```

Encoding categorical data

Label Encoding the "Gender" column

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])
```

```
print(X)
```

```
[[619 'France' 0 ... 1 1 101348.88]
 [608 'Spain' 0 ... 0 1 112542.58]
 [502 'France' 0 ... 1 0 113931.57]
 ...
 [709 'France' 0 ... 0 1 42085.58]
 [772 'Germany' 1 ... 1 0 92888.52]
 [792 'France' 0 ... 1 0 38190.78]]
```

One Hot Encoding the "Geography" column

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

```
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),
[1])], remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

```
print(X)
```

```
[[1.0 0.0 0.0 ... 1 1 101348.88]
 [0.0 0.0 1.0 ... 0 1 112542.58]
 [1.0 0.0 0.0 ... 1 0 113931.57]
 ...
 [1.0 0.0 0.0 ... 0 1 42085.58]
 [0.0 1.0 0.0 ... 1 0 92888.52]
 [1.0 0.0 0.0 ... 1 0 38190.78]]
```

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.3, random_state = 0)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Part 2 - Building the ANN

Initializing the ANN

```
ann = tf.keras.models.Sequential()
```

Adding the input layer and the first hidden layer

```
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

Adding the second hidden layer

```
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

```
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

```
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

Adding the output layer

```
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Part 3 - Training the ANN

Compiling the ANN

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

Training the ANN on the Training set

```
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

```
Epoch 1/100
219/219 [==============================] - 1s 2ms/step - loss: 0.5291
- accuracy: 0.7937
Epoch 2/100
219/219 [==============================] - 0s 2ms/step - loss: 0.4707
- accuracy: 0.7977
Epoch 3/100
219/219 [==============================] - 0s 2ms/step - loss: 0.4503
- accuracy: 0.7977
Epoch 4/100
219/219 [==============================] - 0s 2ms/step - loss: 0.4377
- accuracy: 0.7977
Epoch 5/100
219/219 [==============================] - 0s 2ms/step - loss: 0.4297
- accuracy: 0.7979
Epoch 6/100
219/219 [==============================] - 0s 2ms/step - loss: 0.4224
- accuracy: 0.8033
Epoch 7/100
219/219 [==============================] - 0s 2ms/step - loss: 0.4169
- accuracy: 0.8123
Epoch 8/100
219/219 [==============================] - 0s 1ms/step - loss: 0.4120
- accuracy: 0.8154
Epoch 9/100
219/219 [==============================] - 0s 2ms/step - loss: 0.4080
- accuracy: 0.8193
Epoch 10/100
219/219 [==============================] - 0s 2ms/step - loss: 0.4040
- accuracy: 0.8196
Epoch 11/100
219/219 [==============================] - 0s 2ms/step - loss: 0.4008
- accuracy: 0.8227
Epoch 12/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3979
- accuracy: 0.8219
Epoch 13/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3953
- accuracy: 0.8227
Epoch 14/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3939
- accuracy: 0.8240
Epoch 15/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3913
- accuracy: 0.8257
Epoch 16/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3890
- accuracy: 0.8256
Epoch 17/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3869
```

```
- accuracy: 0.8290
Epoch 18/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3857
- accuracy: 0.8316
Epoch 19/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3842
- accuracy: 0.8309
Epoch 20/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3826
- accuracy: 0.8327
Epoch 21/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3802
- accuracy: 0.8347
Epoch 22/100
219/219 [==============================] - 0s 1ms/step - loss: 0.3800
- accuracy: 0.8344
Epoch 23/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3782
- accuracy: 0.8350
Epoch 24/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3771
- accuracy: 0.8366
Epoch 25/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3752
- accuracy: 0.8380
Epoch 26/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3735
- accuracy: 0.8366
Epoch 27/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3725
- accuracy: 0.8387
Epoch 28/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3711
- accuracy: 0.8399
Epoch 29/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3687
- accuracy: 0.8416
Epoch 30/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3668
- accuracy: 0.8420
Epoch 31/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3646
- accuracy: 0.8433
Epoch 32/100
219/219 [==============================] - 0s 1ms/step - loss: 0.3632
- accuracy: 0.8443
Epoch 33/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3616
- accuracy: 0.8483
Epoch 34/100
```

```
219/219 [==============================] - 0s 2ms/step - loss: 0.3582
- accuracy: 0.8484
Epoch 35/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3558
- accuracy: 0.8546
Epoch 36/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3500
- accuracy: 0.8580
Epoch 37/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3470
- accuracy: 0.8591
Epoch 38/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3433
- accuracy: 0.8609
Epoch 39/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3405
- accuracy: 0.8639
Epoch 40/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3383
- accuracy: 0.8619
Epoch 41/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3390
- accuracy: 0.8629
Epoch 42/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3359
- accuracy: 0.8626
Epoch 43/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3358
- accuracy: 0.8636
Epoch 44/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3347
- accuracy: 0.8636
Epoch 45/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3340
- accuracy: 0.8654
Epoch 46/100
219/219 [==============================] - 0s 1ms/step - loss: 0.3351
- accuracy: 0.8636
Epoch 47/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3333
- accuracy: 0.8650
Epoch 48/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3334
- accuracy: 0.8623
Epoch 49/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3322
- accuracy: 0.8649
Epoch 50/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3327
- accuracy: 0.8641
```

```
Epoch 51/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3317
- accuracy: 0.8654
Epoch 52/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3322
- accuracy: 0.8666
Epoch 53/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3313
- accuracy: 0.8646
Epoch 54/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3306
- accuracy: 0.8650
Epoch 55/100
219/219 [==============================] - 0s 1ms/step - loss: 0.3315
- accuracy: 0.8663
Epoch 56/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3310
- accuracy: 0.8639
Epoch 57/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3312
- accuracy: 0.8620
Epoch 58/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3307
- accuracy: 0.8657
Epoch 59/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3305
- accuracy: 0.8654
Epoch 60/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3298
- accuracy: 0.8663
Epoch 61/100
219/219 [==============================] - 0s 1ms/step - loss: 0.3299
- accuracy: 0.8657
Epoch 62/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3296
- accuracy: 0.8661
Epoch 63/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3282
- accuracy: 0.8657
Epoch 64/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3286
- accuracy: 0.8660
Epoch 65/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3288
- accuracy: 0.8673
Epoch 66/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3283
- accuracy: 0.8679
Epoch 67/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3284
```

```
- accuracy: 0.8670
Epoch 68/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3282
- accuracy: 0.8674
Epoch 69/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3279
- accuracy: 0.8649
Epoch 70/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3281
- accuracy: 0.8673
Epoch 71/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3272
- accuracy: 0.8659
Epoch 72/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3280
- accuracy: 0.8663
Epoch 73/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3275
- accuracy: 0.8669
Epoch 74/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3267
- accuracy: 0.8669
Epoch 75/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3267
- accuracy: 0.8666
Epoch 76/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3269
- accuracy: 0.8700
Epoch 77/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3270
- accuracy: 0.8667
Epoch 78/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3276
- accuracy: 0.8687
Epoch 79/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3268
- accuracy: 0.8661
Epoch 80/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3274
- accuracy: 0.8669
Epoch 81/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3267
- accuracy: 0.8677
Epoch 82/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3265
- accuracy: 0.8669
Epoch 83/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3261
- accuracy: 0.8671
Epoch 84/100
```

```
219/219 [==============================] - 0s 2ms/step - loss: 0.3255
- accuracy: 0.8669
Epoch 85/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3253
- accuracy: 0.8670
Epoch 86/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3256
- accuracy: 0.8677
Epoch 87/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3256
- accuracy: 0.8691
Epoch 88/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3240
- accuracy: 0.8666
Epoch 89/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3256
- accuracy: 0.8691
Epoch 90/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3257
- accuracy: 0.8671
Epoch 91/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3257
- accuracy: 0.8659
Epoch 92/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3257
- accuracy: 0.8663
Epoch 93/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3251
- accuracy: 0.8706
Epoch 94/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3251
- accuracy: 0.8681
Epoch 95/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3251
- accuracy: 0.8696
Epoch 96/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3251
- accuracy: 0.8681
Epoch 97/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3247
- accuracy: 0.8679
Epoch 98/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3251
- accuracy: 0.8660
Epoch 99/100
219/219 [==============================] - 1s 3ms/step - loss: 0.3249
- accuracy: 0.8666
Epoch 100/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3248
- accuracy: 0.8669
```

```
<keras.callbacks.History at 0x7ff86ceb73d0>
```

Part 4 - Making the predictions and evaluating the model

Predicting the result of a single observation

Homework

Use our ANN model to predict if the customer with the following informations will leave the bank:

Geography: France

Credit Score: 600

Gender: Male

Age: 40 years old

Tenure: 3 years

Balance: $ 60000

Number of Products: 2

Does this customer have a credit card? Yes

Is this customer an Active Member: Yes

Estimated Salary: $ 50000

So, should we say goodbye to that customer?

Solution

```
print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])) > 0.5)
```

```
[[False]]
```

Therefore, our ANN model predicts that this customer stays in the bank!

Important note 1: Notice that the values of the features were all input in a double pair of square brackets. That's because the "predict" method always expects a 2D array as the format of its inputs. And putting our values into a double pair of square brackets makes the input exactly a 2D array.

Important note 2: Notice also that the "France" country was not input as a string in the last column but as "1, 0, 0" in the first three columns. That's because of course the predict method expects the one-hot-encoded values of the state, and as we see in the first row of the matrix of features X, "France" was encoded as "1, 0, 0". And be careful to include these values in the first three columns, because the dummy variables are always created in the first columns.

Predicting the Test set results

```
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 1]
 [0 0]
 ...
 [0 0]
 [0 0]
 [1 1]]
```

Making the Confusion Matrix

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[2272  107]
 [ 307  314]]
```

```
0.862
```

Saving the Model

```
ann.save('ann_model')
```

```
from tensorflow import keras
model = keras.models.load_model('ann_model')
```

```
print(model.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2,
1, 1, 50000]])) > 0.5)
```

```
[[False]]
```