artificial_neural_network

November 7, 2024

1 Artificial Neural Network

1.0.1 Importing the libraries

[ ]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
```

[2]:
```
tf.__version__
```

[2]:
```
'2.2.0'
```

1.1 Part 1 - Data Preprocessing

1.1.1 Importing the dataset

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

[4]:
```
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]]
```

[5]:
```
print(y)
```

```
[1 0 1 … 1 1 0]
```

1.1.2 Encoding categorical data

Label Encoding the “Gender” column

[ ]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

```
X[:, 2] = le.fit_transform(X[:, 2])
```

[7]:
```
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))
```

[9]:
```
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]]
```

1.1.3 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 = 0.2,␣
 ↪random_state = 0)
```

1.1.4 Feature Scaling

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

1.2 Part 2 - Building the ANN

1.2.1 Initializing the ANN

[ ]:
```
ann = tf.keras.models.Sequential()
```

1.2.2 Adding the input layer and the first hidden layer

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

1.2.3 Adding the second hidden layer

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

1.2.4 Adding the output layer

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

1.3 Part 3 - Training the ANN

1.3.1 Compiling the ANN

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

1.3.2 Training the ANN on the Training set

[17]:
```
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

```
Epoch 1/100
250/250 [==============================] - 0s 1ms/step - loss: 0.8037 -
accuracy: 0.5185
Epoch 2/100
250/250 [==============================] - 0s 1ms/step - loss: 0.5291 -
accuracy: 0.7901
Epoch 3/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4888 -
accuracy: 0.7952
Epoch 4/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4668 -
accuracy: 0.7979
Epoch 5/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4478 -
accuracy: 0.7994
Epoch 6/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4302 -
accuracy: 0.8049
Epoch 7/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4123 -
```

```
accuracy: 0.8119
Epoch 8/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3947 -
accuracy: 0.8238
Epoch 9/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3807 -
accuracy: 0.8355
Epoch 10/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3720 -
accuracy: 0.8385
Epoch 11/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3664 -
accuracy: 0.8425
Epoch 12/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3629 -
accuracy: 0.8416
Epoch 13/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3599 -
accuracy: 0.8471
Epoch 14/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3580 -
accuracy: 0.8443
Epoch 15/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3564 -
accuracy: 0.8456
Epoch 16/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3550 -
accuracy: 0.8461
Epoch 17/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3539 -
accuracy: 0.8484
Epoch 18/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3530 -
accuracy: 0.8478
Epoch 19/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3521 -
accuracy: 0.8489
Epoch 20/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3509 -
accuracy: 0.8503
Epoch 21/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3498 -
accuracy: 0.8540
Epoch 22/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3491 -
accuracy: 0.8535
Epoch 23/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3481 -
```

```
accuracy: 0.8553
Epoch 24/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3476 -
accuracy: 0.8534
Epoch 25/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3472 -
accuracy: 0.8539
Epoch 26/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3458 -
accuracy: 0.8555
Epoch 27/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3452 -
accuracy: 0.8554
Epoch 28/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3446 -
accuracy: 0.8565
Epoch 29/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3438 -
accuracy: 0.8575
Epoch 30/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3433 -
accuracy: 0.8587
Epoch 31/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3430 -
accuracy: 0.8575
Epoch 32/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3426 -
accuracy: 0.8594
Epoch 33/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3420 -
accuracy: 0.8601
Epoch 34/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3415 -
accuracy: 0.8587
Epoch 35/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3413 -
accuracy: 0.8606
Epoch 36/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3411 -
accuracy: 0.8614
Epoch 37/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3407 -
accuracy: 0.8605
Epoch 38/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3407 -
accuracy: 0.8604
Epoch 39/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3404 -
```

```
accuracy: 0.8601
Epoch 40/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3398 -
accuracy: 0.8605
Epoch 41/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3394 -
accuracy: 0.8626
Epoch 42/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3393 -
accuracy: 0.8621
Epoch 43/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3394 -
accuracy: 0.8604
Epoch 44/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3390 -
accuracy: 0.8611
Epoch 45/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3387 -
accuracy: 0.8627
Epoch 46/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3380 -
accuracy: 0.8610
Epoch 47/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3383 -
accuracy: 0.8611
Epoch 48/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3384 -
accuracy: 0.8608
Epoch 49/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3376 -
accuracy: 0.8611
Epoch 50/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3378 -
accuracy: 0.8631
Epoch 51/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3372 -
accuracy: 0.8619
Epoch 52/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3369 -
accuracy: 0.8624
Epoch 53/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3373 -
accuracy: 0.8620
Epoch 54/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3365 -
accuracy: 0.8635
Epoch 55/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3372 -
```

```
accuracy: 0.8629
Epoch 56/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3364 -
accuracy: 0.8643
Epoch 57/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3364 -
accuracy: 0.8629
Epoch 58/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3361 -
accuracy: 0.8631
Epoch 59/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3359 -
accuracy: 0.8631
Epoch 60/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3356 -
accuracy: 0.8626
Epoch 61/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3356 -
accuracy: 0.8650
Epoch 62/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3354 -
accuracy: 0.8639
Epoch 63/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3352 -
accuracy: 0.8641
Epoch 64/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3344 -
accuracy: 0.8643
Epoch 65/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3354 -
accuracy: 0.8636
Epoch 66/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3348 -
accuracy: 0.8650
Epoch 67/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3346 -
accuracy: 0.8656
Epoch 68/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3348 -
accuracy: 0.8646
Epoch 69/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3341 -
accuracy: 0.8644
Epoch 70/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3341 -
accuracy: 0.8655
Epoch 71/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3341 -
```

```
accuracy: 0.8652
Epoch 72/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3340 -
accuracy: 0.8655
Epoch 73/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3337 -
accuracy: 0.8661
Epoch 74/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3342 -
accuracy: 0.8640
Epoch 75/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3337 -
accuracy: 0.8650
Epoch 76/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3336 -
accuracy: 0.8650
Epoch 77/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3335 -
accuracy: 0.8656
Epoch 78/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3331 -
accuracy: 0.8652
Epoch 79/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3336 -
accuracy: 0.8651
Epoch 80/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3331 -
accuracy: 0.8652
Epoch 81/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3332 -
accuracy: 0.8643
Epoch 82/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3333 -
accuracy: 0.8656
Epoch 83/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3327 -
accuracy: 0.8661
Epoch 84/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3328 -
accuracy: 0.8651
Epoch 85/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3328 -
accuracy: 0.8662
Epoch 86/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3325 -
accuracy: 0.8655
Epoch 87/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3327 -
```

```
accuracy: 0.8654
Epoch 88/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3327 -
accuracy: 0.8645
Epoch 89/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3325 -
accuracy: 0.8674
Epoch 90/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3322 -
accuracy: 0.8655
Epoch 91/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3327 -
accuracy: 0.8650
Epoch 92/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3318 -
accuracy: 0.8650
Epoch 93/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3322 -
accuracy: 0.8635
Epoch 94/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3325 -
accuracy: 0.8650
Epoch 95/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3317 -
accuracy: 0.8662
Epoch 96/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3318 -
accuracy: 0.8646
Epoch 97/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3319 -
accuracy: 0.8649
Epoch 98/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3320 -
accuracy: 0.8641
Epoch 99/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3314 -
accuracy: 0.8648
Epoch 100/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3314 -
accuracy: 0.8660
```

[17]: `<tensorflow.python.keras.callbacks.History at 0x7f8d3ce23978>`

1.4 Part 4 - Making the predictions and evaluating the model

1.4.1 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

[18]:
```
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.

1.4.2 Predicting the Test set results

[19]:
```
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]
 [0 0]]
```

1.4.3 Making the Confusion Matrix

[20]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[1516   79]
 [ 200  205]]
```

[20]:
```
0.8605
```