ANN Implementation

by~ Aadil Mansoori

[ ]:
```
!pip install tensorflow-gpu
```

[2]:
```
import tensorflow as tf
```

[3]:
```
print(tf.__version__)
```

```
2.8.0
```

[4]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

[5]:
```
# Importing the dataset
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:13]
y = dataset.iloc[:, 13]
```

[6]:
```
#Create dummy variables
geography=pd.get_dummies(X["Geography"],drop_first=True)
gender=pd.get_dummies(X['Gender'],drop_first=True)
```

[7]:
```
## Concatenate the Data Frames

X=pd.concat([X,geography,gender],axis=1)

## Drop Unnecessary columns
X=X.drop(['Geography','Gender'],axis=1)

# 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)
```

[8]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[9]:
```
# Part 2 - Now let's make the ANN!
```

[14]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LeakyReLU,PReLU,ELU
from tensorflow.keras.layers import Dropout
```

[15]:
```
# Initialising the ANN
classifier = Sequential()
```

[16]:
```
# Adding the input layer and the first hidden layer
classifier.add(Dense(units=11,activation='relu'))
```

[17]:
```
# Adding the input layer and the first hidden layer
classifier.add(Dense(units=6,activation='relu'))
```

[18]:
```
# Adding the input layer and the first hidden layer
classifier.add(Dense(units=1,activation='relu'))
```

[20]:
```
classifier.
 ↪compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

[22]:
```
model_history=classifier.fit(X_train,y_train,validation_split=0.
 ↪33,batch_size=10,epochs=50)
```

```
Epoch 1/50
536/536 [==============================] - 2s 5ms/step - loss: 0.5597 -
accuracy: 0.7970 - val_loss: 0.5563 - val_accuracy: 0.7974
Epoch 2/50
536/536 [==============================] - 2s 4ms/step - loss: 0.5125 -
accuracy: 0.8022 - val_loss: 0.5131 - val_accuracy: 0.7997
Epoch 3/50
536/536 [==============================] - 3s 6ms/step - loss: 0.4929 -
accuracy: 0.8057 - val_loss: 0.4987 - val_accuracy: 0.8020
Epoch 4/50
536/536 [==============================] - 2s 4ms/step - loss: 0.4816 -
accuracy: 0.8100 - val_loss: 0.4883 - val_accuracy: 0.8080
Epoch 5/50
536/536 [==============================] - 2s 4ms/step - loss: 0.4718 -
accuracy: 0.8125 - val_loss: 0.4482 - val_accuracy: 0.8046
Epoch 6/50
536/536 [==============================] - 2s 4ms/step - loss: 0.4447 -
accuracy: 0.8127 - val_loss: 0.4484 - val_accuracy: 0.8092
Epoch 7/50
```

```
536/536 [==============================] - 2s 4ms/step - loss: 0.3552 -
accuracy: 0.8569 - val_loss: 0.4494 - val_accuracy: 0.8451
Epoch 40/50
536/536 [==============================] - 3s 5ms/step - loss: 0.3573 -
accuracy: 0.8580 - val_loss: 0.4412 - val_accuracy: 0.8497
Epoch 41/50
536/536 [==============================] - 3s 5ms/step - loss: 0.3620 -
accuracy: 0.8582 - val_loss: 0.4336 - val_accuracy: 0.8474
Epoch 42/50
536/536 [==============================] - 2s 3ms/step - loss: 0.3557 -
accuracy: 0.8571 - val_loss: 0.4443 - val_accuracy: 0.8504
Epoch 43/50
536/536 [==============================] - 1s 3ms/step - loss: 0.3514 -
accuracy: 0.8569 - val_loss: 0.4362 - val_accuracy: 0.8512
Epoch 44/50
536/536 [==============================] - 2s 3ms/step - loss: 0.3493 -
accuracy: 0.8559 - val_loss: 0.4511 - val_accuracy: 0.8470
Epoch 45/50
536/536 [==============================] - 1s 2ms/step - loss: 0.3498 -
accuracy: 0.8615 - val_loss: 0.4405 - val_accuracy: 0.8451
Epoch 46/50
536/536 [==============================] - 1s 3ms/step - loss: 0.3502 -
accuracy: 0.8569 - val_loss: 0.4295 - val_accuracy: 0.8501
Epoch 47/50
536/536 [==============================] - 2s 3ms/step - loss: 0.3498 -
accuracy: 0.8574 - val_loss: 0.4436 - val_accuracy: 0.8497
Epoch 48/50
536/536 [==============================] - 2s 3ms/step - loss: 0.3505 -
accuracy: 0.8580 - val_loss: 0.4218 - val_accuracy: 0.8455
Epoch 49/50
536/536 [==============================] - 2s 3ms/step - loss: 0.3556 -
accuracy: 0.8587 - val_loss: 0.4294 - val_accuracy: 0.8470
Epoch 50/50
536/536 [==============================] - 1s 3ms/step - loss: 0.3440 -
accuracy: 0.8587 - val_loss: 0.4300 - val_accuracy: 0.8455
```

[23]:

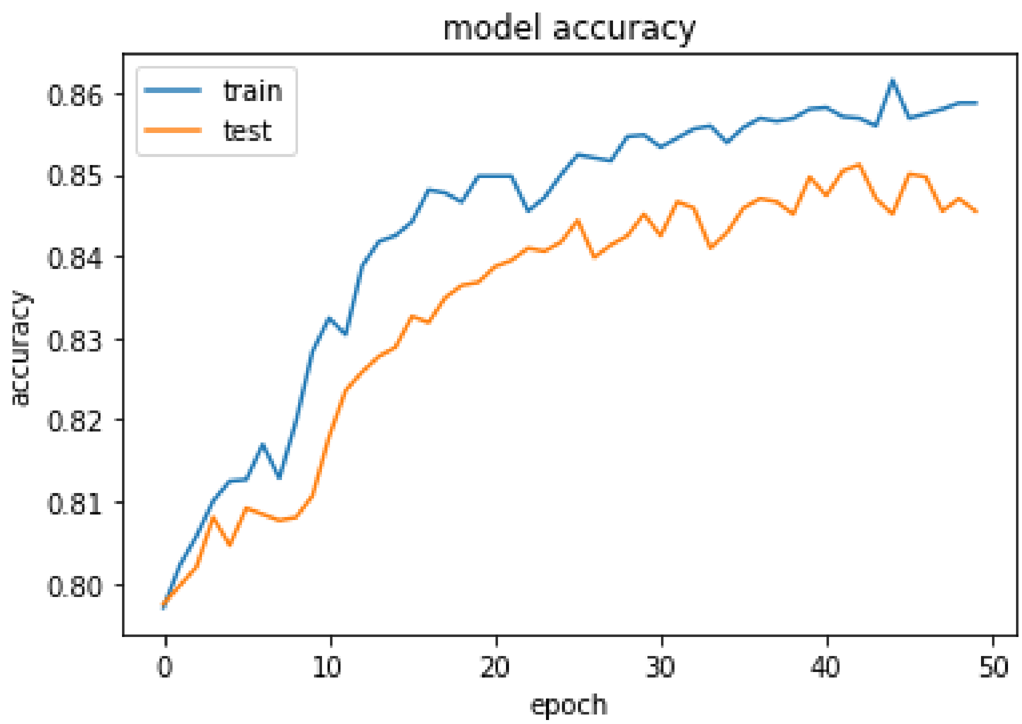
```python
# list all data in history

print(model_history.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

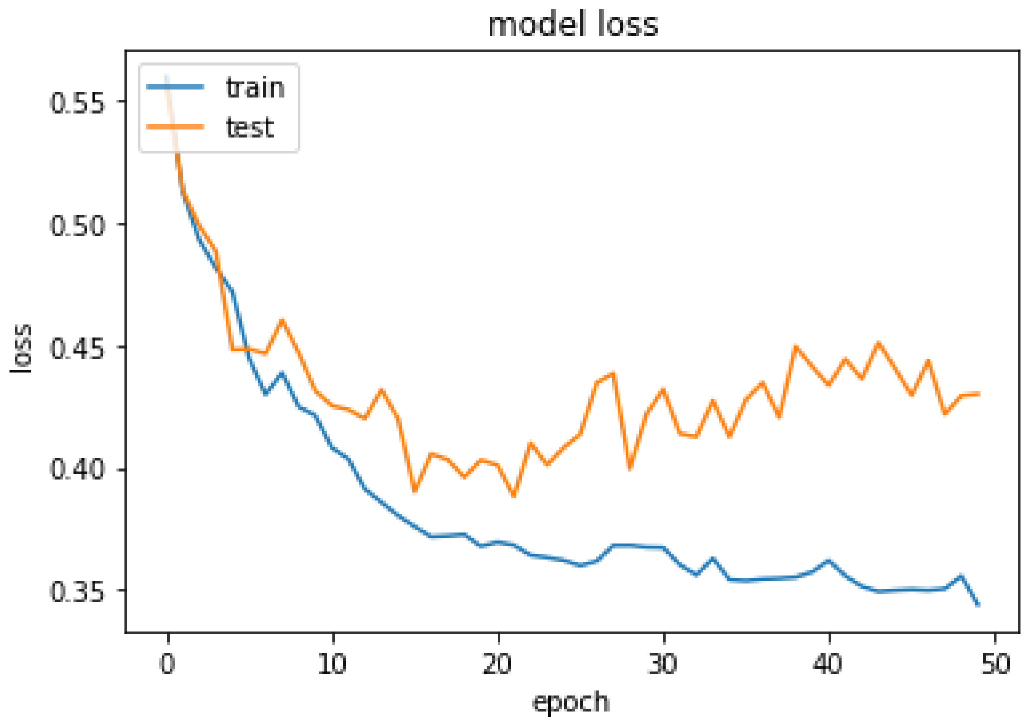
[25]:

```python
# summarize history for accuracy
plt.plot(model_history.history['accuracy'])
plt.plot(model_history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
```

```
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



[26]:
```
# summarize history for loss
plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



[27]:
```
# Part 3 - Making the predictions and evaluating the model

# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)
```

[29]:
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

[29]:
```
array([[1536,   59],
       [ 225,  180]])
```

[30]:
```
# Calculate the Accuracy
from sklearn.metrics import accuracy_score
score=accuracy_score(y_pred,y_test)
```

[31]:
```
score
```

[31]:
```
0.858
```

[ ]:
```

```