Importing Libraries & Dataset

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import keras
import os
```

```
Using TensorFlow backend.
```

In [2]:
```
os.chdir('C:\\Users\\breje\\OneDrive\\Desktop\\ML Dataset\\Deep Learning A-Z\
```

In [3]:
```
df = pd.read_csv('Churn_Modelling.csv')
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
RowNumber          10000 non-null int64
CustomerId         10000 non-null int64
Surname            10000 non-null object
CreditScore        10000 non-null int64
Geography          10000 non-null object
Gender             10000 non-null object
Age                10000 non-null int64
Tenure             10000 non-null int64
Balance            10000 non-null float64
NumOfProducts      10000 non-null int64
HasCrCard          10000 non-null int64
IsActiveMember     10000 non-null int64
EstimatedSalary    10000 non-null float64
Exited             10000 non-null int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

In [5]:
```
df.describe()
```

Out[5]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	N
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	

In [6]:
```
df.head()
```

Out[6]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balar
0	1	15634602	Hargrave	619	France	Female	42	2	0
1	2	15647311	Hill	608	Spain	Female	41	1	83807
2	3	15619304	Onio	502	France	Female	42	8	159660
3	4	15701354	Boni	699	France	Female	39	1	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510

Data Preprocessing

In [7]:
```
df.columns
```

Out[7]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [8]:
```
df = df[['CreditScore', 'Geography',
         'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
         'IsActiveMember', 'EstimatedSalary', 'Exited']]
```

In [9]:
```
df.head()
```

Out[9]:

	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

In [10]:
```
X_geography = pd.DataFrame(pd.get_dummies(df['Geography'], drop_first=True))
```

In [11]:
```
X_gender = pd.DataFrame(pd.get_dummies(df['Gender'], drop_first=True))
```

In [12]:
```
y = pd.DataFrame(df['Exited'])
```

In [13]:
```
y.head()
```

Out[13]:

	Exited
0	1
1	0
2	1
3	0
4	0

In [14]:
```
y.shape
```

Out[14]:
```
(10000, 1)
```

In [15]:
```
X = df.drop(['Geography', 'Gender', 'Exited'], axis=1)
```

In [16]:
```
X.head()
```

Out[16]:

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

In [17]:
```
X.shape
```

Out[17]: (10000, 8)

In [18]:
```
X = pd.concat([X,X_geography,X_gender], axis=1, ignore_index=True, sort=False
```

In [19]:
```
X.head()
```

Out[19]:

	0	1	2	3	4	5	6	7	8	9	10
0	619	42	2	0.00	1	1	1	101348.88	0	0	0
1	608	41	1	83807.86	1	0	1	112542.58	0	1	0
2	502	42	8	159660.80	3	1	0	113931.57	0	0	0
3	699	39	1	0.00	2	0	0	93826.63	0	0	0
4	850	43	2	125510.82	1	1	1	79084.10	0	1	0

In [20]:
```
X.shape
```

Out[20]: (10000, 11)

In [21]:
```
from sklearn.model_selection import train_test_split
```

In [22]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, ranc
```

In [23]:
```
from sklearn.preprocessing import StandardScaler
```

In [24]:
```
scale = StandardScaler()
```

In [25]:
```
X_train = scale.fit_transform(X_train)
```

In [26]:
```
X_test = scale.transform(X_test)
```

In [27]:
```
print(X_train.shape, X_test.shape)
```

```
(9000, 11) (1000, 11)
```

Building an ANN

In [28]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
```

In [29]:
```
model  = Sequential()
```

In [30]:
```
model.add(Dense(input_dim=11, kernel_initializer='uniform', units=6, activati
model.add(Dense(kernel_initializer='uniform', units=6, activation='relu'))
model.add(Dense(kernel_initializer='uniform', units=1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accurac
```

In [31]:
```
model.fit(X_train, y_train, batch_size=10, epochs=50)
```

```
Epoch 1/50
9000/9000 [==============================] - 5s 584us/step - loss: 0.4733 - 
accuracy: 0.7962
Epoch 2/50
9000/9000 [==============================] - 4s 437us/step - loss: 0.4286 - 
accuracy: 0.7963
Epoch 3/50
9000/9000 [==============================] - 4s 432us/step - loss: 0.4252 - 
accuracy: 0.7963
Epoch 4/50
9000/9000 [==============================] - 4s 431us/step - loss: 0.4210 - 
accuracy: 0.8109
Epoch 5/50
9000/9000 [==============================] - 5s 537us/step - loss: 0.4166 - 
accuracy: 0.8231
Epoch 6/50
9000/9000 [==============================] - 4s 459us/step - loss: 0.4138 - 
accuracy: 0.8276
Epoch 7/50
9000/9000 [==============================] - 5s 563us/step - loss: 0.4119 - 
accuracy: 0.8302
Epoch 8/50
9000/9000 [==============================] - 5s 534us/step - loss: 0.4106 - 
accuracy: 0.8327
Epoch 9/50
9000/9000 [==============================] - 4s 500us/step - loss: 0.4094 - 
accuracy: 0.8327
Epoch 10/50
9000/9000 [==============================] - 4s 465us/step - loss: 0.4084 - 
accuracy: 0.8343
Epoch 11/50
9000/9000 [==============================] - 4s 462us/step - loss: 0.4072 - 
accuracy: 0.8334
Epoch 12/50
9000/9000 [==============================] - 4s 464us/step - loss: 0.4067 - 
accuracy: 0.8354
Epoch 13/50
9000/9000 [==============================] - 4s 461us/step - loss: 0.4058 - 
accuracy: 0.8352
Epoch 14/50
9000/9000 [==============================] - 4s 466us/step - loss: 0.4054 - 
accuracy: 0.8366
Epoch 15/50
9000/9000 [==============================] - 5s 549us/step - loss: 0.4044 - 
accuracy: 0.8358
Epoch 16/50
9000/9000 [==============================] - 4s 463us/step - loss: 0.4044 - 
accuracy: 0.8369
Epoch 17/50
9000/9000 [==============================] - 4s 461us/step - loss: 0.4035 - 
accuracy: 0.8358
Epoch 18/50
9000/9000 [==============================] - 4s 470us/step - loss: 0.4033 - 
accuracy: 0.8366
```

```
Epoch 19/50
9000/9000 [==============================] - 4s 468us/step - loss: 0.4032 -
accuracy: 0.8358
Epoch 20/50
9000/9000 [==============================] - 4s 477us/step - loss: 0.4022 -
accuracy: 0.8361
Epoch 21/50
9000/9000 [==============================] - 4s 465us/step - loss: 0.4022 -
accuracy: 0.8368
Epoch 22/50
9000/9000 [==============================] - 4s 478us/step - loss: 0.4016 -
accuracy: 0.8384
Epoch 23/50
9000/9000 [==============================] - 4s 467us/step - loss: 0.4017 -
accuracy: 0.8367
Epoch 24/50
9000/9000 [==============================] - 4s 466us/step - loss: 0.4009 -
accuracy: 0.8367
Epoch 25/50
9000/9000 [==============================] - 4s 469us/step - loss: 0.4006 -
accuracy: 0.8377
Epoch 26/50
9000/9000 [==============================] - 4s 467us/step - loss: 0.4007 -
accuracy: 0.8353
Epoch 27/50
9000/9000 [==============================] - 4s 470us/step - loss: 0.4005 -
accuracy: 0.8377
Epoch 28/50
9000/9000 [==============================] - 4s 469us/step - loss: 0.4001 -
accuracy: 0.8371
Epoch 29/50
9000/9000 [==============================] - 4s 470us/step - loss: 0.4005 -
accuracy: 0.8364
Epoch 30/50
9000/9000 [==============================] - 4s 468us/step - loss: 0.4001 -
accuracy: 0.8373
Epoch 31/50
9000/9000 [==============================] - 4s 467us/step - loss: 0.3997 -
accuracy: 0.8366
Epoch 32/50
9000/9000 [==============================] - 4s 467us/step - loss: 0.3999 -
accuracy: 0.8371
Epoch 33/50
9000/9000 [==============================] - 4s 469us/step - loss: 0.3995 -
accuracy: 0.8370
Epoch 34/50
9000/9000 [==============================] - 4s 467us/step - loss: 0.3994 -
accuracy: 0.8356
Epoch 35/50
9000/9000 [==============================] - 4s 469us/step - loss: 0.3995 -
accuracy: 0.8372
Epoch 36/50
9000/9000 [==============================] - 4s 483us/step - loss: 0.3992 -
accuracy: 0.8371
Epoch 37/50
9000/9000 [==============================] - 5s 506us/step - loss: 0.3985 -
accuracy: 0.8377
```

```
Epoch 38/50
9000/9000 [==============================] - 4s 496us/step - loss: 0.3988 -
accuracy: 0.8367
Epoch 39/50
9000/9000 [==============================] - 4s 490us/step - loss: 0.3987 -
accuracy: 0.8356
Epoch 40/50
9000/9000 [==============================] - 4s 487us/step - loss: 0.3984 -
accuracy: 0.8358
Epoch 41/50
9000/9000 [==============================] - 5s 509us/step - loss: 0.3985 -
accuracy: 0.8370
Epoch 42/50
9000/9000 [==============================] - 4s 494us/step - loss: 0.3984 -
accuracy: 0.8368
Epoch 43/50
9000/9000 [==============================] - 4s 494us/step - loss: 0.3982 -
accuracy: 0.8368
Epoch 44/50
9000/9000 [==============================] - 5s 506us/step - loss: 0.3984 -
accuracy: 0.8371
Epoch 45/50
9000/9000 [==============================] - 5s 546us/step - loss: 0.3983 -
accuracy: 0.8373
Epoch 46/50
9000/9000 [==============================] - 5s 508us/step - loss: 0.3982 -
accuracy: 0.8358
Epoch 47/50
9000/9000 [==============================] - 5s 501us/step - loss: 0.3980 -
accuracy: 0.8381
Epoch 48/50
9000/9000 [==============================] - 5s 507us/step - loss: 0.3981 -
accuracy: 0.8371
Epoch 49/50
9000/9000 [==============================] - 5s 503us/step - loss: 0.3983 -
accuracy: 0.8383
Epoch 50/50
9000/9000 [==============================] - 4s 492us/step - loss: 0.3978 -
accuracy: 0.8369
```

Out[31]:
```
<keras.callbacks.callbacks.History at 0x27e6226a388>
```

Evaluating the model

In [32]:
```
y_pred = model.predict(X_test)
```

In [33]:
```
y_pred = y_pred >0.50
```

In [34]:
```
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score
```

In [35]:
```
confusion_matrix(y_test, y_pred)
```

Out[35]:
```
array([[775,  21],
       [147,  57]], dtype=int64)
```

In [36]:
```
f1_score(y_test, y_pred)
```

Out[36]:
```
0.4042553191489362
```

In [37]:
```
accuracy_score(y_test, y_pred)
```

Out[37]:
```
0.832
```

* 83% accuracy! Not bad eh? (false positive is too high though)Now let's implement regularization through CV & Droput; then improve the accuracy using Parameter tuning*

Cross Validation

In [38]:
```
from keras.wrappers.scikit_learn import KerasClassifier
```

In [39]:
```
from sklearn.model_selection import cross_val_score
```

In [40]:
```
def classifier():
    model  = Sequential()
    model.add(Dense(input_dim=11, kernel_initializer='uniform', units=6, acti
    model.add(Dense(kernel_initializer='uniform', units=6, activation='relu')
    model.add(Dense(kernel_initializer='uniform', units=1, activation='sigmoi
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc
    return model
```

In [41]:
```
CV_model = KerasClassifier(build_fn=classifier, batch_size=10, epochs=50)
```

In [42]:
```
cv_score = cross_val_score(estimator=CV_model, X=X_train, y=y_train, cv=10, r
```

In [43]:
```
cv_score.mean()
```

Out[43]:
```
0.8406666696071625
```

In [44]:
```
cv_score.std()
```

Out[44]:
```
0.012193903836001005
```

Dropout

In [45]:
```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
```

In [46]:
```
def classifier():
    model  = Sequential()
    model.add(Dense(input_dim=11, kernel_initializer='uniform', units=6, acti
    model.add(Dropout(p=0.1))
    model.add(Dense(kernel_initializer='uniform', units=6, activation='relu')
    model.add(Dropout(p=0.1))
    model.add(Dense(kernel_initializer='uniform', units=1, activation='sigmoi
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc
    return model
```

In [47]:
```
CV_model = KerasClassifier(build_fn=classifier, batch_size=10, epochs=50)
cv_score = cross_val_score(estimator=CV_model, X=X_train, y=y_train, cv=10, r
```

```
C:\Users\breje\AppData\Local\Continuum\anaconda3\lib\site-packages\joblib\e
xternals\loky\process_executor.py:706: UserWarning: A worker stopped while
some jobs were given to the executor. This can be caused by a too short wor
ker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
```

In [48]:
```
cv_score.mean()
```

Out[48]: 0.8376666665077209

In [49]:
```
cv_score.std()
```

Out[49]: 0.010959525774749282

Parameter Tuning

In [54]:
```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
```

In [55]:
```
def classifier(optimizer):
    model  = Sequential()
    model.add(Dense(input_dim=11, kernel_initializer='uniform', units=6, acti
    model.add(Dense(kernel_initializer='uniform', units=6, activation='relu')
    model.add(Dense(kernel_initializer='uniform', units=1, activation='sigmoi
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['
    return model
```

In [56]:
```
model = KerasClassifier(build_fn=classifier)
parameters = {'batch_size':[5, 20],
              'nb_epoch':[75, 100],
              'optimizer':['adam', 'rmsprop']}
gs_model = GridSearchCV(estimator=model, param_grid=parameters, scoring='accu
```

In [57]:
```
gs = gs_model.fit(X_train, y_train)
```

```
- accuracy: 0.7948
Epoch 1/1
8100/8100 [==============================] - 13s 2ms/step - loss: 0.4721
- accuracy: 0.7962
Epoch 1/1
8100/8100 [==============================] - ETA: 0s - loss: 0.4630 - ac
curacy: 0.79 - 13s 2ms/step - loss: 0.4631 - accuracy: 0.7970
Epoch 1/1
8100/8100 [==============================] - 13s 2ms/step - loss: 0.4739
- accuracy: 0.7983
Epoch 1/1
8100/8100 [==============================] - 13s 2ms/step - loss: 0.4735
- accuracy: 0.7956
Epoch 1/1
8100/8100 [==============================] - 13s 2ms/step - loss: 0.4786
- accuracy: 0.7975
Epoch 1/1
8100/8100 [==============================] - 13s 2ms/step - loss: 0.4711
- accuracy: 0.7964
```

In [58]:
```
gs.best_params_
gs.best_score_
```

Out[58]: 0.8001111111111111

In [ ]: