In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

In [4]:
```
data = pd.read_csv("Churn_Modelling.csv", index_col='RowNumber')
data.head()
```

Out[4]:

RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	Nu
1	15634602	Hargrave	619	France	Female	42	2	0.00	
2	15647311	Hill	608	Spain	Female	41	1	83807.86	
3	15619304	Onio	502	France	Female	42	8	159660.80	
4	15701354	Boni	699	France	Female	39	1	0.00	
5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	

In [5]:
```
df = data.copy()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 1 to 10000
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   CustomerId       10000 non-null  int64  
 1   Surname          10000 non-null  object 
 2   CreditScore      10000 non-null  int64  
 3   Geography        10000 non-null  object 
 4   Gender           10000 non-null  object 
 5   Age              10000 non-null  int64  
 6   Tenure           10000 non-null  int64  
 7   Balance          10000 non-null  float64
 8   NumOfProducts    10000 non-null  int64  
 9   HasCrCard        10000 non-null  int64  
 10  IsActiveMember   10000 non-null  int64  
 11  EstimatedSalary  10000 non-null  float64
 12  Exited           10000 non-null  int64  
dtypes: float64(2), int64(8), object(3)
memory usage: 1.1+ MB
```

In [6]:
```
X_columns = df.columns.tolist()[2:12]
y_columns = df.columns.tolist()[-1:]
print(f'All columns: {df.columns.tolist()}')
print(f'X values: {X_columns}')
print(f'y values: {y_columns}')
```

```
All columns: ['CustomerId', 'Surname', 'CreditScore', 'Geography', 'Gender', 'A
ge', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'Esti
matedSalary', 'Exited']
X values: ['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance', 'N
umOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']
y values: ['Exited']
```

In [7]:
```
X = df[X_columns].values
y = df[y_columns].values
```

In [8]:
```
print("Original:", X[:8,1])
from sklearn.preprocessing import LabelEncoder

label_X_country_encoder = LabelEncoder()
X[:,1] = label_X_country_encoder.fit_transform(X[:,1])
print("Encoded: " ,X[:8,1])
```

```
Original: ['France' 'Spain' 'France' 'France' 'Spain' 'Spain' 'France' 'German
y']
Encoded:  [0 2 0 0 2 2 0 1]
```

In [9]:
```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline


pipeline = Pipeline(
    [('Categorizer', ColumnTransformer(
         [ # Gender
          ("Gender Label encoder", OneHotEncoder(categories='auto', drop='first')
           # Geography
          ("Geography One Hot", OneHotEncoder(categories='auto', drop='first'), [
         ], remainder='passthrough', n_jobs=1)),
     # Standard Scaler for the classifier
    ('Normalizer', StandardScaler())
    ])
```

In [10]:
```
X = pipeline.fit_transform(X)
```

In [11]:
```
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_
```

In [12]:
```
print(f'training shapes: {X_train.shape}, {y_train.shape}')
print(f'testing shapes: {X_test.shape}, {y_test.shape}')
```

```
training shapes: (8000, 11), (8000, 1)
testing shapes: (2000, 11), (2000, 1)
```

In [13]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout

classifier = Sequential()
```

In [14]:
```
# This adds the input layer (by specifying input dimension) AND the first hidden
classifier.add(Dense(6, activation = 'relu', input_shape = (X_train.shape[1], )))
classifier.add(Dropout(rate=0.1))
```

In [15]:
```
# Adding the second hidden layer
# Notice that we do not need to specify input dim.
classifier.add(Dense(6, activation = 'relu'))
classifier.add(Dropout(rate=0.1))
```

In [16]:
```
# Adding the output layer
# Notice that we do not need to specify input dim.
# we have an output of 1 node, which is the the desired dimensions of our output
# We use the sigmoid because we want probability outcomes
classifier.add(Dense(1, activation = 'sigmoid'))
```

In [17]:
```
classifier.summary()
```

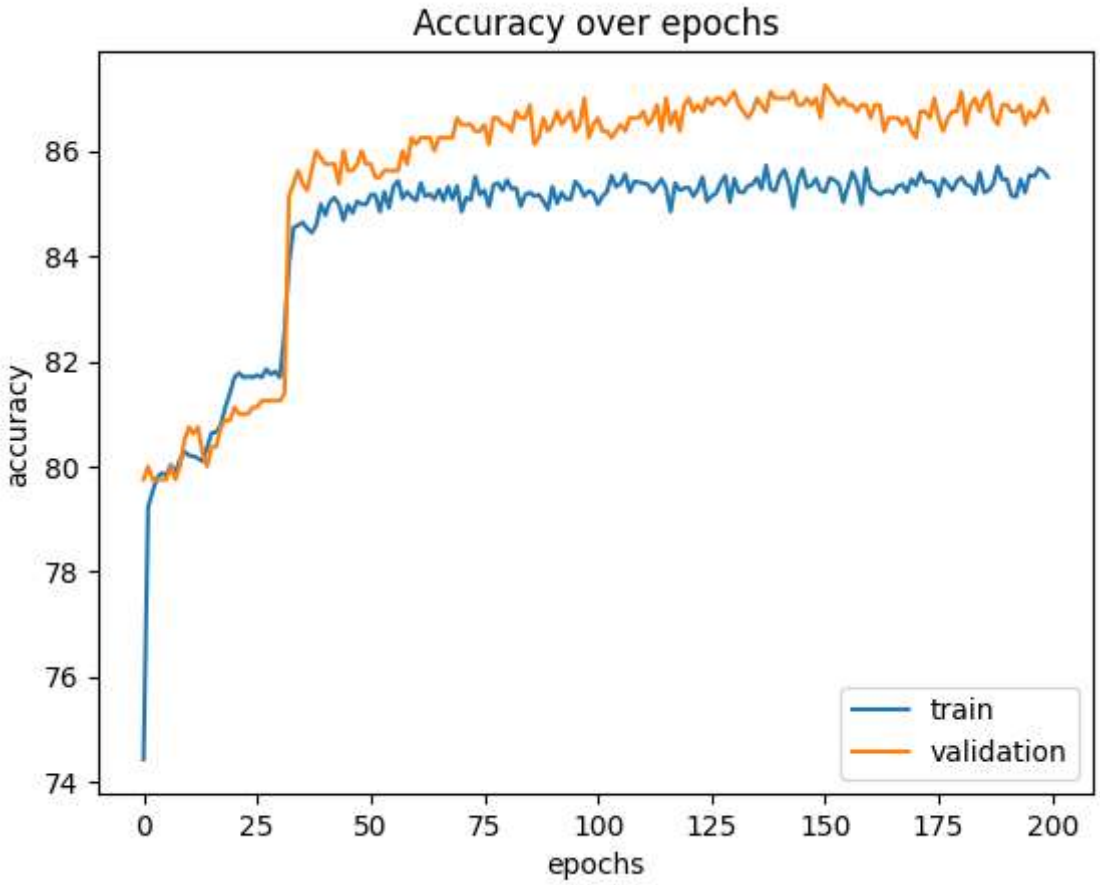
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 6)                 72

 dropout (Dropout)           (None, 6)                 0

 dense_1 (Dense)             (None, 6)                 42

 dropout_1 (Dropout)         (None, 6)                 0

 dense_2 (Dense)             (None, 1)                 7

=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
```

In [18]:
```
classifier.compile(optimizer='adam', loss = 'binary_crossentropy', metrics=['accu
```

In [19]:
```
history = classifier.fit(X_train, y_train, batch_size=32, epochs=200, validation_
```

```
acy: 0.8675 - 184ms/epoch - 818us/step
Epoch 136/200
225/225 - 0s - loss: 0.3711 - accuracy: 0.8549 - val_loss: 0.3370 - val_accur
acy: 0.8700 - 201ms/epoch - 893us/step
Epoch 137/200
225/225 - 0s - loss: 0.3777 - accuracy: 0.8528 - val_loss: 0.3378 - val_accur
acy: 0.8687 - 181ms/epoch - 806us/step
Epoch 138/200
225/225 - 0s - loss: 0.3751 - accuracy: 0.8572 - val_loss: 0.3365 - val_accur
acy: 0.8675 - 198ms/epoch - 880us/step
Epoch 139/200
225/225 - 0s - loss: 0.3752 - accuracy: 0.8528 - val_loss: 0.3388 - val_accur
acy: 0.8712 - 191ms/epoch - 851us/step
Epoch 140/200
225/225 - 0s - loss: 0.3763 - accuracy: 0.8524 - val_loss: 0.3370 - val_accur
acy: 0.8700 - 201ms/epoch - 892us/step
Epoch 141/200
225/225 - 0s - loss: 0.3693 - accuracy: 0.8551 - val_loss: 0.3349 - val_accur
acy: 0.8700 - 198ms/epoch - 880us/step
Epoch 142/200
```

In [20]:
```
plt.plot(np.array(history.history['accuracy']) * 100)
plt.plot(np.array(history.history['val_accuracy']) * 100)
plt.ylabel('accuracy')
plt.xlabel('epochs')
plt.legend(['train', 'validation'])
plt.title('Accuracy over epochs')
plt.show()
```



In [21]:
```
y_pred = classifier.predict(X_test)
print(y_pred[:5])
```

```
63/63 [==============================] - 0s 664us/step
[[0.34736758]
 [0.37604278]
 [0.26283768]
 [0.09331284]
 [0.1003078 ]]
```

In [22]:
```
y_pred = (y_pred > 0.5).astype(int)
print(y_pred[:5])
```

```
[[0]
 [0]
 [0]
 [0]
 [0]]
```

In [23]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[23]:
```
array([[1529,   66],
       [ 205,  200]], dtype=int64)
```

In [24]:
```
print(((cm[0][0]+cm[1][1])*100)/(len(y_test)), '% of testing data was classified
```

```
86.45 % of testing data was classified correctly
```

In [ ]: