Training and Testing a Neural Network to predict the output of clinical trials on patients who experience and who who did not experience side effects.

In [121…

```python
import numpy as np
from random import randint
from sklearn.utils import shuffle
from sklearn.preprocessing import MinMaxScaler
```

In [122…

```python
train_labels=[]
train_samples=[]
```

In [123…

```python
for i in range (50):
     # 5 % of younger population who did experience side effects
    random_younger =randint (13,64)
    train_samples.append(random_younger)
    train_labels.append(1)

    #5% of older population who did not experience side effects
    random_older =randint (65,100)
    train_samples.append(random_older)
    train_labels.append(0)

for i in range (1000):
     # 95 % of younger population who did not experience side effects
    random_younger =randint (13,64)
    train_samples.append(random_younger)
    train_labels.append(0)


     # 95 % of older population who did experience side effects
    random_older =randint (65,100)
    train_samples.append(random_older)
    train_labels.append(1)
```

In [ ]:

In [124…

```python
train_labels = np.array(train_labels)
train_samples = np .array(train_samples)
train_labels ,train_samples =shuffle(train_labels ,train_samples)
```

In [125…

```python
scalar = MinMaxScaler(feature_range=(0,1))
scaled_train_samples= scalar.fit_transform(train_samples.reshape(-1,1))
```

In [126…

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation ,Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
```

In [127…

```python
model=Sequential([
   Dense(units=16 ,input_shape=(1,),activation='relu' ),
   Dense(units=32 ,activation='relu' ),
   Dense(units=2 ,activation='softmax' )

])
```

In [128…

```python
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_15 (Dense)             (None, 16)                32
_________________________________________________________________
dense_16 (Dense)             (None, 32)                544
_________________________________________________________________
dense_17 (Dense)             (None, 2)                 66
=================================================================
Total params: 642
Trainable params: 642
Non-trainable params: 0
_________________________________________________________________
```

In [129…

```python
model.compile(optimizer=Adam(learning_rate=0.0001), loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

In [130…

```python
#model.fit(x=scaled_train_samples, y=train_labels, batch_size=10,epochs=30,shuffle=True,verbose=2)
```

In [131…

```python
model.fit(x=scaled_train_samples, y=train_labels,validation_split=0.1, batch_size=10,epochs=30,shuffle=True,verbose=2)
```

```
Epoch 1/30
189/189 - 1s - loss: 0.6669 - accuracy: 0.5640 - val_loss: 0.6486 - val_accuracy: 0.6619
Epoch 2/30
189/189 - 0s - loss: 0.6402 - accuracy: 0.6286 - val_loss: 0.6216 - val_accuracy: 0.7048
Epoch 3/30
189/189 - 0s - loss: 0.6152 - accuracy: 0.6735 - val_loss: 0.5948 - val_accuracy: 0.7333
Epoch 4/30
189/189 - 0s - loss: 0.5901 - accuracy: 0.6968 - val_loss: 0.5697 - val_accuracy: 0.7524
Epoch 5/30
189/189 - 0s - loss: 0.5647 - accuracy: 0.7386 - val_loss: 0.5444 - val_accuracy: 0.7714
Epoch 6/30
189/189 - 0s - loss: 0.5389 - accuracy: 0.7725 - val_loss: 0.5193 - val_accuracy: 0.7905
Epoch 7/30
189/189 - 0s - loss: 0.5131 - accuracy: 0.7952 - val_loss: 0.4955 - val_accuracy: 0.8095
Epoch 8/30
189/189 - 0s - loss: 0.4880 - accuracy: 0.8275 - val_loss: 0.4710 - val_accuracy: 0.8143
Epoch 9/30
189/189 - 0s - loss: 0.4618 - accuracy: 0.8481 - val_loss: 0.4453 - val_accuracy: 0.8190
Epoch 10/30
189/189 - 0s - loss: 0.4363 - accuracy: 0.8656 - val_loss: 0.4225 - val_accuracy: 0.8381
Epoch 11/30
189/189 - 0s - loss: 0.4129 - accuracy: 0.8751 - val_loss: 0.4025 - val_accuracy: 0.8667
Epoch 12/30
189/189 - 0s - loss: 0.3917 - accuracy: 0.8952 - val_loss: 0.3837 - val_accuracy: 0.8810
Epoch 13/30
189/189 - 0s - loss: 0.3725 - accuracy: 0.9037 - val_loss: 0.3671 - val_accuracy: 0.8810
Epoch 14/30
189/189 - 0s - loss: 0.3557 - accuracy: 0.9069 - val_loss: 0.3531 - val_accuracy: 0.9048
Epoch 15/30
189/189 - 0s - loss: 0.3410 - accuracy: 0.9169 - val_loss: 0.3410 - val_accuracy: 0.9048
Epoch 16/30
189/189 - 0s - loss: 0.3287 - accuracy: 0.9201 - val_loss: 0.3305 - val_accuracy: 0.9095
Epoch 17/30
189/189 - 0s - loss: 0.3177 - accuracy: 0.9243 - val_loss: 0.3220 - val_accuracy: 0.9095
Epoch 18/30
189/189 - 0s - loss: 0.3083 - accuracy: 0.9270 - val_loss: 0.3138 - val_accuracy: 0.9095
Epoch 19/30
189/189 - 0s - loss: 0.3002 - accuracy: 0.9296 - val_loss: 0.3074 - val_accuracy: 0.9095
Epoch 20/30
189/189 - 0s - loss: 0.2933 - accuracy: 0.9312 - val_loss: 0.3020 - val_accuracy: 0.9333
Epoch 21/30
189/189 - 0s - loss: 0.2875 - accuracy: 0.9323 - val_loss: 0.2981 - val_accuracy: 0.9333
Epoch 22/30
189/189 - 0s - loss: 0.2826 - accuracy: 0.9344 - val_loss: 0.2949 - val_accuracy: 0.9333
Epoch 23/30
189/189 - 0s - loss: 0.2784 - accuracy: 0.9360 - val_loss: 0.2909 - val_accuracy: 0.9333
Epoch 24/30
189/189 - 0s - loss: 0.2747 - accuracy: 0.9344 - val_loss: 0.2887 - val_accuracy: 0.9333
Epoch 25/30
189/189 - 0s - loss: 0.2715 - accuracy: 0.9397 - val_loss: 0.2864 - val_accuracy: 0.9333
Epoch 26/30
189/189 - 0s - loss: 0.2687 - accuracy: 0.9360 - val_loss: 0.2846 - val_accuracy: 0.9333
Epoch 27/30
189/189 - 0s - loss: 0.2663 - accuracy: 0.9407 - val_loss: 0.2829 - val_accuracy: 0.9333
Epoch 28/30
189/189 - 0s - loss: 0.2640 - accuracy: 0.9349 - val_loss: 0.2817 - val_accuracy: 0.9333
Epoch 29/30
189/189 - 0s - loss: 0.2619 - accuracy: 0.9407 - val_loss: 0.2804 - val_accuracy: 0.9333
Epoch 30/30
189/189 - 0s - loss: 0.2601 - accuracy: 0.9407 - val_loss: 0.2789 - val_accuracy: 0.9333
```

Out[131…

```
<tensorflow.python.keras.callbacks.History at 0x1ac6e3e7880>
```

In [132…

```python
test_labels=[]
test_samples=[]
for i in range (50):
     # 5 % of younger population who did experience side effects
    random_younger =randint (13,64)
    test_samples.append(random_younger)
    test_labels.append(1)

    #5% of older population who did not experience side effects
    random_older =randint (65,100)
    test_samples.append(random_older)
    test_labels.append(0)

for i in range (1000):
     # 95 % of younger population who did not experience side effects
    random_younger =randint (13,64)
    test_samples.append(random_younger)
    test_labels.append(0)


     # 95 % of older population who did experience side effects
    random_older =randint (65,100)
    test_samples.append(random_older)
    test_labels.append(1)
```

In [133…

```python
test_labels = np.array(test_labels)
test_samples = np .array(test_samples)
test_labels ,test_samples =shuffle(test_labels ,test_samples)
```

In [134…

```python
scaled_test_samples= scalar.fit_transform(test_samples.reshape(-1,1))
```

In [135…

```python
predictions=model.predict(x=scaled_test_samples,batch_size=10,verbose=0)
```

In [136…

```python
rounded_predictions=np.argmax(predictions,axis=1)
```

In [137…

```python
%matplotlib inline
from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt
```

In [138…

```python
cm =confusion_matrix(y_true=test_labels, y_pred=rounded_predictions)
```

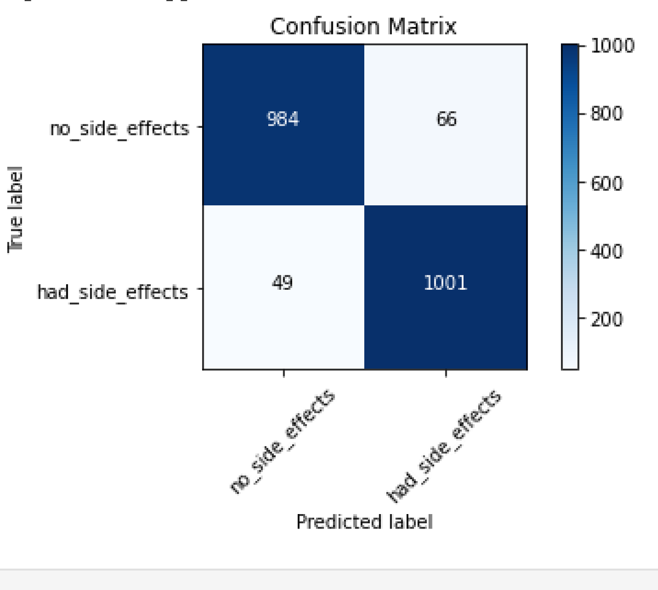
In [139…

```python
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [140…

```python
cm_plot_labels=['no_side_effects', 'had_side_effects']
plot_confusion_matrix(cm=cm, classes=cm_plot_labels,title='Confusion Matrix')
```

```
Confusion matrix, without normalization
[[ 984   66]
 [  49 1001]]
```



In [ ]: