```
#Step 1 – Importing libraries required for Fake news Classifier.


import re
import nltk
import numpy as np
import pandas as pd
import tensorflow as tf
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import one_hot
from sklearn.metrics import confusion_matrix,accuracy_score
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding,LSTM,Dense,Dropout
from sklearn.metrics import confusion_matrix

nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

Step 2 – Reading input data.

```
df = pd.read_csv('/content/train.csv.zip')
df.dropna(inplace=True)
df.reset_index(inplace=True)
df.head(10)
```

	index	id	title	author	text	label
0	0	0	House Dem Aide: We Didn't Even See Comey's Let...	Darrell Lucus	House Dem Aide: We Didn't Even See Comey's Let...	1
1	1	1	FLYNN: Hillary Clinton, Big Woman on Campus - ...	Daniel J. Flynn	Ever get the feeling your life circles the rou...	0
2	2	2	Why the Truth Might Get You Fired	Consortiumnews.com	Why the Truth Might Get You Fired October 29, ...	1
3	3	3	15 Civilians Killed In Single US Airstrike Hav...	Jessica Purkiss	Videos 15 Civilians Killed In Single US Airstr...	1
4	4	4	Iranian woman jailed for fictional unpublished...	Howard Portnoy	Print \nAn Iranian woman has been sentenced to...	1
5	5	5	Jackie Mason: Hollywood Would Love Trump if He...	Daniel Nussbaum	In these trying times, Jackie Mason is the Voi...	0
6	7	7	Benoît Hamon Wins French Socialist Party's Pre...	Alissa J. Rubin	PARIS — France chose an idealistic, traditi...	0
7	9	9	A Back-Channel Plan for Ukraine and Russia, Co...	Megan Twohey and Scott Shane	A week before Michael T. Flynn resigned as nat...	0
8	10	10	Obama's Organizing for Action Partners with So...	Aaron Klein	Organizing for Action, the activist group that...	0
9	11	11	BBC Comedy Sketch "Real Housewives of ISIS" Ca...	Chris Tomlinson	The BBC produced spoof on the "Real Housewives...	0

Next steps: Generate code with df View recommended plots New interactive sheet

Step 3 – Creating X and y data.

```
X = df['title']
y = df['label']
```

For X we are just taking the title column. For y we are just taking the label column.

Step 4 – Cleaning input data.

```
ps = PorterStemmer()
corpus = []

for i in range(len(X)):
    text = X[i]
    text = re.sub('[^a-zA-Z]',' ',text)
    text = text.lower()
    text = text.split()
    text = [ps.stem(t) for t in text if t not in stopwords.words('english')]
    corpus.append(' '.join(text))
```

Here we are traversing in X and then just simply using regex to clean our data and store it in the corpus list.

First of all, we are just replacing everything that is not an alphabet with a space.

Then we are lowercasing it and splitting it.

Then we are checking if the words are not in stopwords, then stem it.

Simply join these results and make a sentence out of them and add it to the corpus list.

Step 5 – Encoding input data.

```
vocab_size = 5000
sent_len = 20

one_hot_encoded = [one_hot(x,vocab_size) for x in corpus]
one_hot_encoded = pad_sequences(one_hot_encoded,maxlen=sent_len)
one_hot_encoded[0]
```

```
array([   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,  873,
       2629, 2297, 2248, 1241, 1112, 3180, 3929, 3827, 2974], dtype=int32)
```

Here we are encoding our text data to numerical data using one_hot. Remember this one hot is not that 0s and 1s. In this one-hot encoding, we assign a random number using hashing to the word. The random word is chosen from the range 0-vocab_size. Then we are padding the sequences with 0s to make every line of the same length. And then simply we are checking how our first sentence looks like after these 2 operations.

Step 6 – Processing X and y data.

```
X = np.array(one_hot_encoded)
y = np.array(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Converting X and y to NumPy arrays and simply splitting the data using traintestsplit.

Step 7 – Creating the model.

```
no_of_output_features = 40

model = Sequential()

model.add(Embedding(vocab_size,no_of_output_features,input_length=sent_len))
model.add(Dropout(0.5))
model.add(LSTM(100))
model.add(Dropout(0.5))
model.add(Dense(1))

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_2 (Embedding)     (None, 20, 40)            200000    
                                                                 
 dropout_4 (Dropout)         (None, 20, 40)            0         
                                                                 
 lstm_2 (LSTM)               (None, 100)               56400     
                                                                 
 dropout_5 (Dropout)         (None, 100)               0         
                                                                 
 dense_2 (Dense)             (None, 1)                 101       
                                                                 
=================================================================
Total params: 256501 (1001.96 KB)
Trainable params: 256501 (1001.96 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Here we are creating our model. Our model has just 4 layers. The first layer is the Embedding layer which will convert the number array which we saw above into a vector of 40 dimensions followed by a Dropout layer. And then we have an LSTM layer with 100 nodes followed by a Dropout layer. Dropout layers are for preventing Overfitting.

```
no_of_output_features = 40

model = Sequential()
model.add(Embedding(vocab_size,no_of_output_features,input_length=sent_len))
model.add(Dropout(0.5))
model.add(LSTM(100))
model.add(Dropout(0.5))
model.add(Dense(1))

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_3 (Embedding)     (None, 20, 40)            200000    
                                                                 
 dropout_6 (Dropout)         (None, 20, 40)            0         
                                                                 
 lstm_3 (LSTM)               (None, 100)               56400     
                                                                 
 dropout_7 (Dropout)         (None, 100)               0         
                                                                 
 dense_3 (Dense)             (None, 1)                 101       
                                                                 
=================================================================
Total params: 256501 (1001.96 KB)
Trainable params: 256501 (1001.96 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Step 8 – Training the Fake news Classifier model.

```
model.fit(X_train,y_train,validation_data=(X_test,y_test),batch_size=64,epochs=40)
```

```
Epoch 1/40
192/192 [==============================] - 12s 49ms/step - loss: 0.5178 - accuracy: 0.7531 - val_loss: 0.3308 - val_accuracy: 0.9012
Epoch 2/40
192/192 [==============================] - 7s 37ms/step - loss: 0.2622 - accuracy: 0.9008 - val_loss: 0.2950 - val_accuracy: 0.8978
Epoch 3/40
192/192 [==============================] - 12s 61ms/step - loss: 0.2318 - accuracy: 0.9297 - val_loss: 0.3775 - val_accuracy: 0.9016
Epoch 4/40
192/192 [==============================] - 9s 47ms/step - loss: 0.2656 - accuracy: 0.8887 - val_loss: 0.3749 - val_accuracy: 0.8880
Epoch 5/40
192/192 [==============================] - 7s 39ms/step - loss: 0.1949 - accuracy: 0.9328 - val_loss: 0.4334 - val_accuracy: 0.8896
Epoch 6/40
192/192 [==============================] - 9s 49ms/step - loss: 0.1899 - accuracy: 0.9479 - val_loss: 0.5687 - val_accuracy: 0.9059
Epoch 7/40
192/192 [==============================] - 9s 46ms/step - loss: 0.1669 - accuracy: 0.9609 - val_loss: 0.4977 - val_accuracy: 0.9027
Epoch 8/40
192/192 [==============================] - 13s 69ms/step - loss: 0.1806 - accuracy: 0.9424 - val_loss: 0.5379 - val_accuracy: 0.8943
Epoch 9/40
```

```
192/192 [==============================] - 9s 45ms/step - loss: 0.1453 - accuracy: 0.9660 - val_loss: 0.5742 - val_accuracy: 0.9069
Epoch 10/40
192/192 [==============================] - 9s 49ms/step - loss: 0.1341 - accuracy: 0.9704 - val_loss: 0.5792 - val_accuracy: 0.9075
Epoch 11/40
192/192 [==============================] - 7s 39ms/step - loss: 0.2116 - accuracy: 0.9375 - val_loss: 0.7016 - val_accuracy: 0.8963
Epoch 12/40
192/192 [==============================] - 9s 48ms/step - loss: 0.1264 - accuracy: 0.9728 - val_loss: 0.7026 - val_accuracy: 0.9077
Epoch 13/40
192/192 [==============================] - 9s 48ms/step - loss: 0.1089 - accuracy: 0.9755 - val_loss: 0.6689 - val_accuracy: 0.9024
Epoch 14/40
192/192 [==============================] - 9s 47ms/step - loss: 0.1053 - accuracy: 0.9795 - val_loss: 0.7232 - val_accuracy: 0.8983
Epoch 15/40
192/192 [==============================] - 9s 49ms/step - loss: 0.0939 - accuracy: 0.9850 - val_loss: 0.7462 - val_accuracy: 0.8920
Epoch 16/40
192/192 [==============================] - 9s 47ms/step - loss: 0.0974 - accuracy: 0.9776 - val_loss: 0.7224 - val_accuracy: 0.8926
Epoch 17/40
192/192 [==============================] - 8s 41ms/step - loss: 0.0806 - accuracy: 0.9887 - val_loss: 0.7452 - val_accuracy: 0.9011
Epoch 18/40
192/192 [==============================] - 10s 50ms/step - loss: 0.0746 - accuracy: 0.9883 - val_loss: 0.7486 - val_accuracy: 0.9125
Epoch 19/40
192/192 [==============================] - 10s 52ms/step - loss: 0.0705 - accuracy: 0.9914 - val_loss: 0.7849 - val_accuracy: 0.9109
Epoch 20/40
192/192 [==============================] - 10s 50ms/step - loss: 0.0643 - accuracy: 0.9917 - val_loss: 0.8070 - val_accuracy: 0.9062
Epoch 21/40
192/192 [==============================] - 9s 48ms/step - loss: 0.0571 - accuracy: 0.9932 - val_loss: 0.8603 - val_accuracy: 0.9150
Epoch 22/40
192/192 [==============================] - 7s 38ms/step - loss: 0.0585 - accuracy: 0.9928 - val_loss: 0.8558 - val_accuracy: 0.9097
Epoch 23/40
192/192 [==============================] - 10s 50ms/step - loss: 0.0567 - accuracy: 0.9942 - val_loss: 0.8713 - val_accuracy: 0.9112
Epoch 24/40
192/192 [==============================] - 10s 50ms/step - loss: 0.0716 - accuracy: 0.9898 - val_loss: 0.8874 - val_accuracy: 0.9046
Epoch 25/40
192/192 [==============================] - 7s 39ms/step - loss: 0.0575 - accuracy: 0.9935 - val_loss: 0.8517 - val_accuracy: 0.9087
Epoch 26/40
192/192 [==============================] - 9s 48ms/step - loss: 0.0519 - accuracy: 0.9936 - val_loss: 0.8851 - val_accuracy: 0.9044
Epoch 27/40
192/192 [==============================] - 8s 41ms/step - loss: 0.0491 - accuracy: 0.9949 - val_loss: 0.9136 - val_accuracy: 0.9094
Epoch 28/40
192/192 [==============================] - 9s 45ms/step - loss: 0.0455 - accuracy: 0.9952 - val_loss: 0.9568 - val_accuracy: 0.9057
Epoch 29/40
```

Step 9 – Checking metrics of the model.Checking the accuracy of the Fake news Classifier model.

```
from sklearn.metrics import confusion_matrix, accuracy_score

# Make predictions
y_pred = (model.predict(X_test) > 0.5).astype("int32")

# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score:", accuracy)
```

```
189/189 [==============================] - 2s 12ms/step
Confusion Matrix:
[[3106  313]
 [ 267 2349]]
Accuracy Score: 0.903893951946976
```