EXPERIMENT-9

AIM: **To apply Recurrent Neural Models for text classification.**

CODE and OUTPUT:

Dataset Link: https://www.kaggle.com/uciml/sms-spam-collection-dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding
from keras.optimizers import RMSprop
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping
%matplotlib inline
```

```
df = pd.read_csv('spam.csv',delimiter=',',encoding='latin-1')
df.head()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

```
df.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'],axis=1,inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   v1      5572 non-null   object
 1   v2      5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

```
df.head()
```

	v1	v2
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

```
X = df.v2
Y = df.v1
le = LabelEncoder()
Y = le.fit_transform(Y)
print(Y)
print(Y.shape)
Y = Y.reshape(-1,1)
```

```
[0 0 1 ... 0 0 0]
(5572,)
```

```
print(Y.shape)
```

```
(5572, 1)
```

```
#Split into training and test data.

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.15)
```

Process the data

1. Tokenize the data and convert the text to sequences.
2. Add padding to ensure that all the sequences have the same shape.
3. There are many ways of taking the max_len and here an arbitrary length of 150 is chosen.

```
max_words = 1000
max_len = 150
tok = Tokenizer(num_words=max_words)
tok.fit_on_texts(X_train)
sequences = tok.texts_to_sequences(X_train)
sequences_matrix = sequence.pad_sequences(sequences,maxlen=max_len)
```

LSTM

```
def RNN():
    inputs = Input(name='inputs',shape=[max_len])
    layer = Embedding(max_words,50,input_length=max_len)(inputs)
    layer = LSTM(64)(layer)
    layer = Dense(256,name='FC1')(layer)
    layer = Activation('relu')(layer)
    layer = Dropout(0.5)(layer)
    layer = Dense(1,name='out_layer')(layer)
    layer = Activation('sigmoid')(layer)
    model = Model(inputs=inputs,outputs=layer)
    return model
```

```
model = RNN()
model.summary()
model.compile(loss='binary_crossentropy',optimizer=RMSprop(),metrics=['accuracy'])
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inputs (InputLayer)          [(None, 150)]             0
_________________________________________________________________
embedding (Embedding)        (None, 150, 50)           50000
_________________________________________________________________
lstm (LSTM)                  (None, 64)                29440
_________________________________________________________________
FC1 (Dense)                  (None, 256)               16640
_________________________________________________________________
activation (Activation)      (None, 256)               0
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
out_layer (Dense)            (None, 1)                 257
_________________________________________________________________
activation_1 (Activation)    (None, 1)                 0
=================================================================
Total params: 96,337
Trainable params: 96,337
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(sequences_matrix,Y_train,batch_size=128,epochs=10,
          validation_split=0.2,callbacks=[EarlyStopping(monitor='val_loss')])
```

```
Epoch 1/10
30/30 [==============================] - 9s 208ms/step - loss: 0.4471 - accuracy: 0.8222 - val_loss: 0.1592 - val_accuracy: 0.9462
Epoch 2/10
30/30 [==============================] - 6s 184ms/step - loss: 0.1153 - accuracy: 0.9742 - val_loss: 0.0697 - val_accuracy: 0.9810
Epoch 3/10
30/30 [==============================] - 5s 181ms/step - loss: 0.0468 - accuracy: 0.9876 - val_loss: 0.0680 - val_accuracy: 0.9800
Epoch 4/10
30/30 [==============================] - 6s 185ms/step - loss: 0.0276 - accuracy: 0.9908 - val_loss: 0.0607 - val_accuracy: 0.9821
Epoch 5/10
30/30 [==============================] - 5s 181ms/step - loss: 0.0260 - accuracy: 0.9910 - val_loss: 0.0615 - val_accuracy: 0.9800
<tensorflow.python.keras.callbacks.History at 0x7fec98caf1d0>
```

```
#Process the test set data.
```

```
test_sequences = tok.texts_to_sequences(X_test)
test_sequences_matrix = sequence.pad_sequences(test_sequences,maxlen=max_len)
```

```
#Evaluate the model on the test set.
accr = model.evaluate(test_sequences_matrix,Y_test)
```

```
27/27 [==============================] - 1s 20ms/step - loss: 0.0490 - accuracy: 0.9856
```

```
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))
```

```
Test set
  Loss: 0.049
  Accuracy: 0.986
```