```
# import libraries for reading data, exploring and plotting
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
%matplotlib inline
# library for train test split
from sklearn.model_selection import train_test_split
# deep learning libraries for text pre-processing
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
# Modeling
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GlobalAveragePooling1D, Den
from sklearn.model_selection import train_test_split
from keras.layers import Dense , LSTM , Embedding , Dropout , Activation ,
from sklearn.preprocessing import LabelEncoder
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.preprocessing import sequence
from tensorflow.keras.utils import to_categorical
from keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
url = 'https://raw.githubusercontent.com/ShresthaSudip/SMS_Spam_Detection_
messages = pd.read_csv(url, sep ='\t',names=["label", "message"])
messages[:3]
```

	label	message
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...

```
messages.describe()
```

	label	message
count	5572	5572

```
duplicatedRow = messages[messages.duplicated()]
print(duplicatedRow[:5])
```

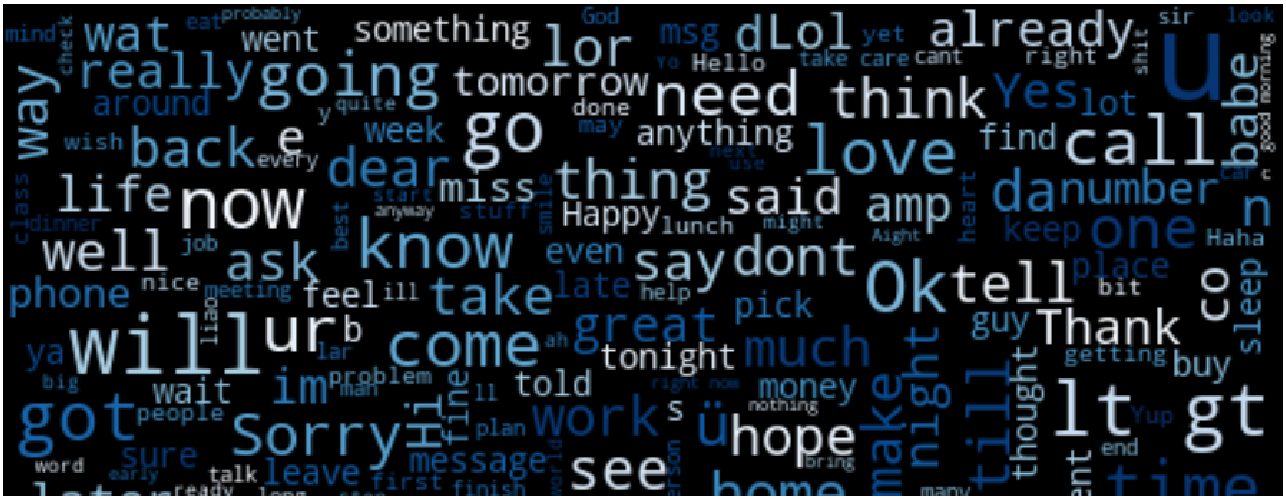
```
    label                                            message
103   ham  As per your request 'Melle Melle (Oru Minnamin...
154   ham  As per your request 'Melle Melle (Oru Minnamin...
207   ham  As I entered my cabin my PA said, '' Happy B'd...
223   ham                              Sorry, I'll call later
326   ham                   No calls..messages..missed calls
```

```
messages.groupby('label').describe().T
```

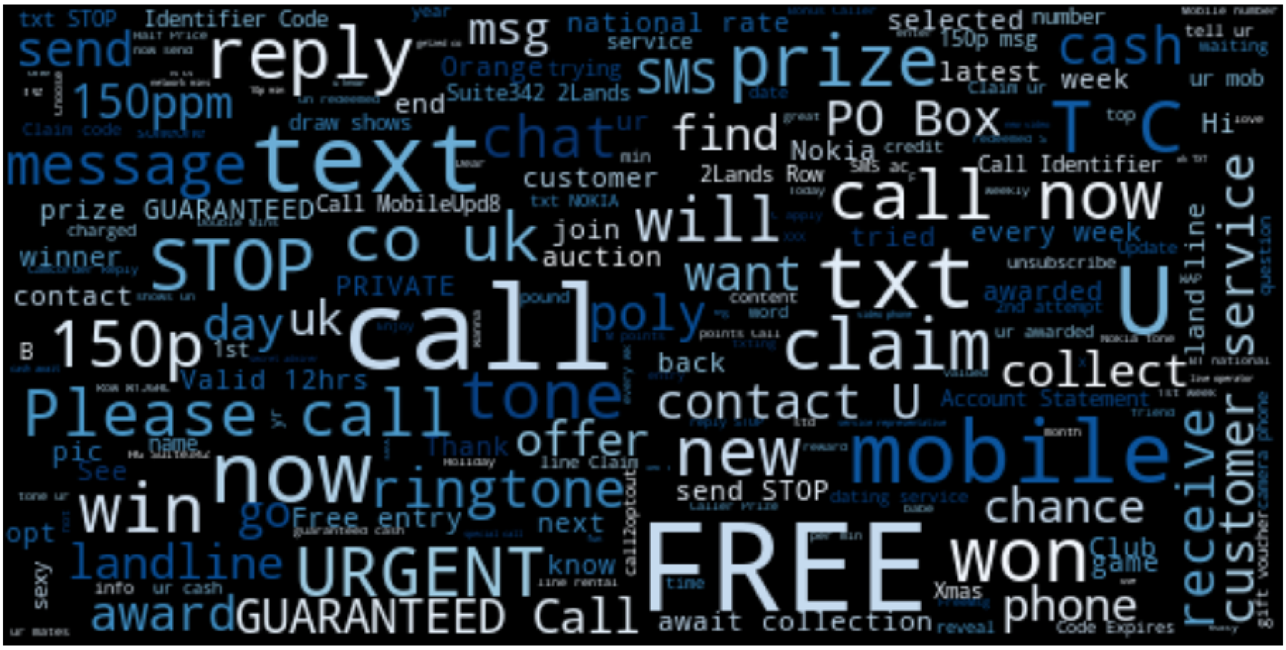
	label	ham	spam
message	**count**	4825	747
	unique	4516	653
	top	Sorry, I'll call later	Please call our customer service representativ...
	freq	30	4

```
# Get all the ham and spam emails
ham_msg = messages[messages.label =='ham']
spam_msg = messages[messages.label=='spam']
# Create numpy list to visualize using wordcloud
ham_msg_text = " ".join(ham_msg.message.to_numpy().tolist())
spam_msg_text = " ".join(spam_msg.message.to_numpy().tolist())
```

```
# wordcloud of ham messages
ham_msg_cloud = WordCloud(width =520, height =260, stopwords=STOPWORDS,max_
plt.figure(figsize=(16,10))
plt.imshow(ham_msg_cloud, interpolation='bilinear')
plt.axis('off') # turn off axis
plt.show()
```

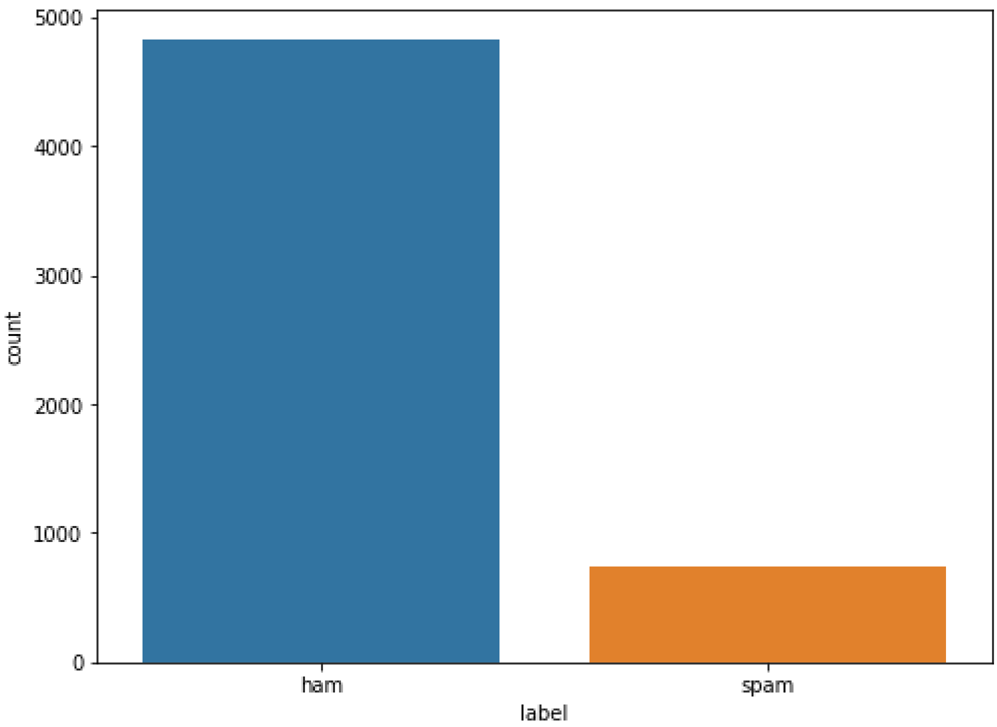



```
# wordcloud of spam messages
spam_msg_cloud = WordCloud(width =520, height =260, stopwords=STOPWORDS,ma
plt.figure(figsize=(16,10))
plt.imshow(spam_msg_cloud, interpolation='bilinear')
plt.axis('off') # turn off axis
plt.show()
```




```
# we can observe imbalance data here
plt.figure(figsize=(8,6))
sns.countplot(messages.label)
# Percentage of spam messages
(len(spam_msg)/len(ham_msg))*100 # 15.48%
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pas:
  FutureWarning
15.481865284974095
```

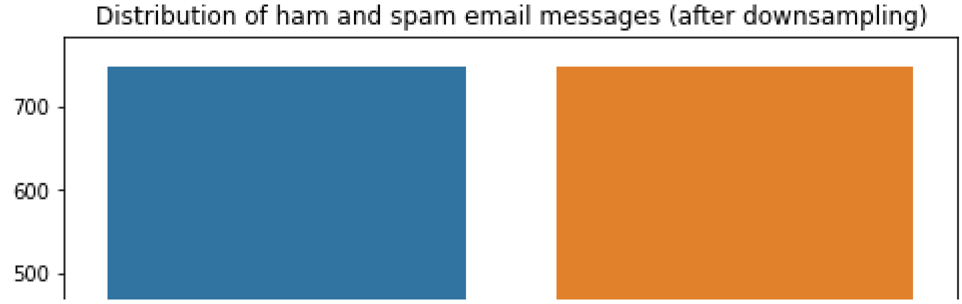



```
# one way to fix it is to downsample the ham msg
ham_msg_df = ham_msg.sample(n = len(spam_msg), random_state = 44)
spam_msg_df = spam_msg
print(ham_msg_df.shape, spam_msg_df.shape)
```

```
(747, 2) (747, 2)
```

```
# Create a dataframe with these ham and spam msg
msg_df = ham_msg_df.append(spam_msg_df).reset_index(drop=True)
plt.figure(figsize=(8,6))
sns.countplot(msg_df.label)
plt.title('Distribution of ham and spam email messages (after downsampling
plt.xlabel('Message types')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
Text(0.5, 0, 'Message types')
```




```
# Get length column for each text
msg_df['text_length'] = msg_df['message'].apply(len)
#Calculate average length by label types
labels = msg_df.groupby('label').mean()
labels
```

	text_length
label	
ham	73.238286
spam	138.670683

```
# Map ham label as 0 and spam as 1
msg_df['msg_type']= msg_df['label'].map({'ham': 0, 'spam': 1})
msg_label = msg_df['msg_type'].values
# Split data into train and test
train_msg, test_msg, train_labels, test_labels = train_test_split(msg_df['

# Defining pre-processing hyperparameters
max_len = 50
trunc_type = "post"
padding_type = "post"
oov_tok = "<OOV>"
vocab_size = 500

tokenizer = Tokenizer(num_words = vocab_size, char_level=False, oov_token =
tokenizer.fit_on_texts(train_msg)

# Get the word_index
word_index = tokenizer.word_index
word_index
```

```
 'dload': 944,
 'noline': 945,
 'rentl': 946,
 'netcollex': 947,
 'hard': 948,
```

```
 'making': 949,
 'energy': 950,
 'calls£1': 951,
 'profit': 952,
 'st': 953,
 'always': 954,
 "'": 955,
 'never': 956,
 'ppm': 957,
 'company': 958,
 'tel': 959,
 'spook': 960,
 'put': 961,
 'country': 962,
 'ntwk': 963,
 '08001950382': 964,
 'tenerife': 965,
 '£900': 966,
 'email': 967,
 '8552': 968,
 'callertune': 969,
 'weight': 970,
 'thats': 971,
 'textpod': 972,
 'happiness': 973,
 'blue': 974,
 '1000s': 975,
 '80082': 976,
 'lifetime': 977,
 'greet': 978,
 'mean': 979,
 '0808': 980,
 '145': 981,
 '4742': 982,
 '9am': 983,
 '11pm': 984,
 'pod': 985,
 'deliveredtomorrow': 986,
 '80488': 987,
 'found': 988,
 'subs': 989,
 '05': 990,
 'project': 991,
 'exciting': 992,
 'xy': 993,
 '84025': 994,
 '21': 995,
 'voicemail': 996,
 '4u': 997,
 'title': 998,
 'titles': 999,
 'babes': 1000,
 ...}
```

```
# check how many words
tot_words = len(word_index)
print('There are %s unique tokens in training data. ' % tot_words)
```

```
There are 4169 unique tokens in training data.
```

```
# Sequencing and padding on training and testing
training_sequences = tokenizer.texts_to_sequences(train_msg)
training_padded = pad_sequences (training_sequences, maxlen = max_len, padd
testing_sequences = tokenizer.texts_to_sequences(test_msg)
testing_padded = pad_sequences(testing_sequences, maxlen = max_len,
padding = padding_type, truncating = trunc_type)
```

```
# Shape of train tensor
print('Shape of training tensor: ', training_padded.shape)
print('Shape of testing tensor: ', testing_padded.shape)
```

```
Shape of training tensor:  (1195, 50)
Shape of testing tensor:  (299, 50)
```

```
# Before padding
len(training_sequences[0]), len(training_sequences[1])

# After padding
len(training_padded[0]), len(training_padded[1])
```

```
(50, 50)
```

```
print(training_padded[0])
```

```
[  1  47 186   9  34   1   3  24   1   2 274   2   7 152 275 135  34  10
  15   6   7  34 274  85  15  17   1   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
```

Dense Hidden Layer

```
vocab_size = 500 # As defined earlier
embeding_dim = 16
drop_value = 0.2 # dropout
n_dense = 24
```

```
#Dense model architecture
model = Sequential()
model.add(Embedding(vocab_size, embeding_dim, input_length=max_len))
model.add(GlobalAveragePooling1D())
model.add(Dense(24, activation='relu'))
```

```
model.add(Dropout(drop_value))
model.add(Dense(1, activation='sigmoid'))
```

```
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_8 (Embedding)     (None, 50, 16)            8000      
                                                                 
 global_average_pooling1d_2   (None, 16)               0         
 (GlobalAveragePooling1D)                                        
                                                                 
 dense_14 (Dense)            (None, 24)                408       
                                                                 
 dropout_3 (Dropout)         (None, 24)                0         
                                                                 
 dense_15 (Dense)            (None, 1)                 25        
                                                                 
=================================================================
Total params: 8,433
Trainable params: 8,433
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='binary_crossentropy',optimizer='adam' ,metrics=['accur
```

```
# fitting a dense spam detector model
num_epochs = 30
early_stop = EarlyStopping(monitor='val_loss', patience=3)
history = model.fit(training_padded, train_labels, epochs=num_epochs, vali
```

```
Epoch 1/30
38/38 - 1s - loss: 0.6843 - accuracy: 0.7598 - val_loss: 0.6748 - val_accuracy: 0.792
Epoch 2/30
38/38 - 0s - loss: 0.6549 - accuracy: 0.8510 - val_loss: 0.6327 - val_accuracy: 0.829
Epoch 3/30
38/38 - 0s - loss: 0.5922 - accuracy: 0.8711 - val_loss: 0.5572 - val_accuracy: 0.846
Epoch 4/30
38/38 - 0s - loss: 0.4998 - accuracy: 0.8787 - val_loss: 0.4642 - val_accuracy: 0.852
Epoch 5/30
38/38 - 0s - loss: 0.3993 - accuracy: 0.8987 - val_loss: 0.3790 - val_accuracy: 0.866
Epoch 6/30
38/38 - 0s - loss: 0.3161 - accuracy: 0.9079 - val_loss: 0.3139 - val_accuracy: 0.889
Epoch 7/30
38/38 - 0s - loss: 0.2566 - accuracy: 0.9163 - val_loss: 0.2679 - val_accuracy: 0.893
Epoch 8/30
38/38 - 0s - loss: 0.2141 - accuracy: 0.9389 - val_loss: 0.2255 - val_accuracy: 0.906
Epoch 9/30
38/38 - 0s - loss: 0.1877 - accuracy: 0.9397 - val_loss: 0.2063 - val_accuracy: 0.916
Epoch 10/30
38/38 - 0s - loss: 0.1625 - accuracy: 0.9456 - val_loss: 0.1757 - val_accuracy: 0.926
Epoch 11/30
38/38 - 0s - loss: 0.1450 - accuracy: 0.9573 - val_loss: 0.1748 - val_accuracy: 0.933
Epoch 12/30
```

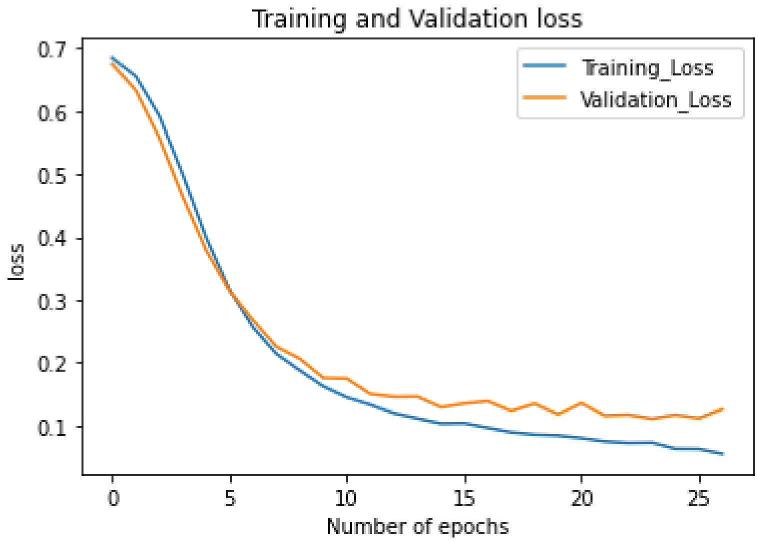
```
38/38 - 0s - loss: 0.1335 - accuracy: 0.9598 - val_loss: 0.1504 - val_accuracy: 0.946
Epoch 13/30
38/38 - 0s - loss: 0.1194 - accuracy: 0.9632 - val_loss: 0.1458 - val_accuracy: 0.943
Epoch 14/30
38/38 - 0s - loss: 0.1112 - accuracy: 0.9649 - val_loss: 0.1462 - val_accuracy: 0.936
Epoch 15/30
38/38 - 0s - loss: 0.1031 - accuracy: 0.9715 - val_loss: 0.1295 - val_accuracy: 0.949
Epoch 16/30
38/38 - 0s - loss: 0.1036 - accuracy: 0.9674 - val_loss: 0.1353 - val_accuracy: 0.939
Epoch 17/30
38/38 - 0s - loss: 0.0964 - accuracy: 0.9724 - val_loss: 0.1390 - val_accuracy: 0.943
Epoch 18/30
38/38 - 0s - loss: 0.0893 - accuracy: 0.9699 - val_loss: 0.1236 - val_accuracy: 0.946
Epoch 19/30
38/38 - 0s - loss: 0.0856 - accuracy: 0.9715 - val_loss: 0.1352 - val_accuracy: 0.939
Epoch 20/30
38/38 - 0s - loss: 0.0842 - accuracy: 0.9707 - val_loss: 0.1178 - val_accuracy: 0.946
Epoch 21/30
38/38 - 0s - loss: 0.0802 - accuracy: 0.9715 - val_loss: 0.1360 - val_accuracy: 0.943
Epoch 22/30
38/38 - 0s - loss: 0.0748 - accuracy: 0.9766 - val_loss: 0.1157 - val_accuracy: 0.946
Epoch 23/30
38/38 - 0s - loss: 0.0724 - accuracy: 0.9741 - val_loss: 0.1171 - val_accuracy: 0.946
Epoch 24/30
38/38 - 0s - loss: 0.0730 - accuracy: 0.9732 - val_loss: 0.1107 - val_accuracy: 0.953
Epoch 25/30
38/38 - 0s - loss: 0.0632 - accuracy: 0.9791 - val_loss: 0.1169 - val_accuracy: 0.943
Epoch 26/30
38/38 - 0s - loss: 0.0629 - accuracy: 0.9791 - val_loss: 0.1115 - val_accuracy: 0.949
Epoch 27/30
38/38 - 0s - loss: 0.0552 - accuracy: 0.9791 - val_loss: 0.1262 - val_accuracy: 0.936
```

```
# Model performance on test data
model.evaluate(testing_padded, test_labels)
```

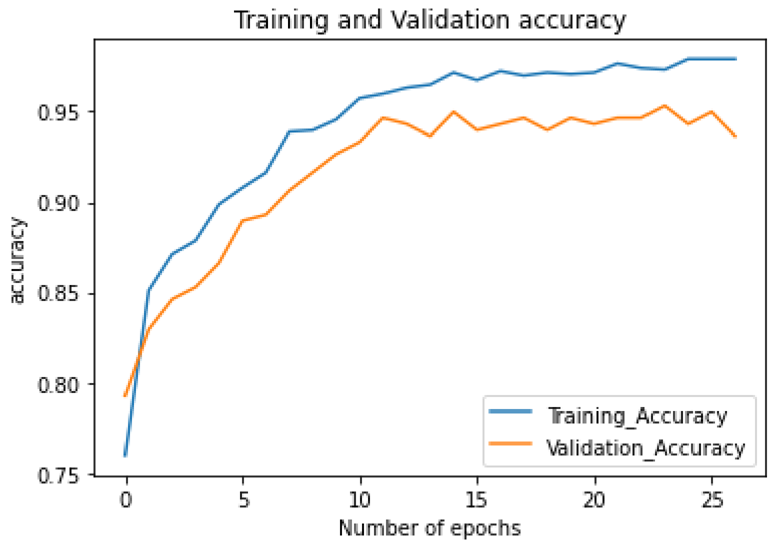
```
10/10 [==============================] - 0s 2ms/step - loss: 0.1262 - accuracy: 0.936
[0.12617890536785126, 0.9364548325538635]
```

```
# Read as a dataframe
metrics = pd.DataFrame(history.history)
# Rename column
metrics.rename(columns = {'loss': 'Training_Loss', 'accuracy': 'Training_A
def plot_graphs1(var1, var2, string):
    metrics[[var1, var2]].plot()
    plt.title('Training and Validation ' + string)
    plt.xlabel ('Number of epochs')
    plt.ylabel(string)
    plt.legend([var1, var2])


plot_graphs1('Training_Loss', 'Validation_Loss', 'loss')
```




```
plot_graphs1('Training_Accuracy', 'Validation_Accuracy', 'accuracy')
```



Long Short Term Memory (LSTM) Model

```
#LSTM hyperparameters
n_lstm = 20
drop_lstm =0.2
```

```
#LSTM Spam detection architecture
#LSTM Spam detection architecture
model1 = Sequential()
model1.add(Embedding(vocab_size, embeding_dim, input_length=max_len))
model1.add(LSTM(n_lstm, dropout=drop_lstm, return_sequences=True))
model1.add(LSTM(n_lstm, dropout=drop_lstm, return_sequences=True))
model1.add(Dense(1, activation='sigmoid'))
```

```
messages
```

	label	message
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...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

```
messages.describe().T
```

	count	unique	top	freq
label	5572	2	ham	4825
message	5572	5169	Sorry, I'll call later	30

```
messages.shape
```

```
(5572, 2)
```

```
messages.isnull().sum()
```

```
label      0
message    0
dtype: int64
```

```
sns.countplot(messages.label)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pas
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f079e124ed0>
```


```
X = messages.message
Y = messages.label
le = LabelEncoder()
Y = le.fit_transform(Y)
```



```
Y = Y.reshape(-1,1)
```

```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.2)
```

```
max_word = 1000
max_len = 250
token = Tokenizer(num_words = max_word)
token.fit_on_texts(X_train)
sequences = token.texts_to_sequences(X_train)
seq_matrix = pad_sequences(sequences , maxlen = max_len)
```

```
model = Sequential()
model.add(Embedding(max_word , 32 , input_length = max_len))
model.add(LSTM(64))
model.add(Flatten())

model.add(Dense(250, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(120, activation='relu'))

model.add(Dense(1, activation='sigmoid'))


# Compile the model
model.compile(loss = 'binary_crossentropy' , optimizer = 'RMSprop' , metri
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_10 (Embedding)    (None, 250, 32)           32000
```

```
 lstm_13 (LSTM)              (None, 64)                24832

 flatten_1 (Flatten)         (None, 64)                0

 dense_17 (Dense)            (None, 250)               16250

 dropout_4 (Dropout)         (None, 250)               0

 dense_18 (Dense)            (None, 120)               30120

 dense_19 (Dense)            (None, 1)                 121

=================================================================
Total params: 103,323
Trainable params: 103,323
Non-trainable params: 0
_________________________________________________________________
```

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

```
Epoch 1/10
28/28 [==============================] - 15s 421ms/step - loss: 0.3305 - accuracy: 0
Epoch 2/10
28/28 [==============================] - 11s 398ms/step - loss: 0.0754 - accuracy: 0
```

```
test_seq = token.texts_to_sequences(X_test)
test_seq_matrix = pad_sequences(test_seq,maxlen=max_len)
```

```
scores = model.evaluate(test_seq_matrix, Y_test, verbose=0)
scores
```

```
[0.05998490750789642, 0.9838564991950989]
```

```
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 98.39%
```

Colab paid products - Cancel contracts here

0s completed at 4:07 PM