Spam Message Classification using LSTM

1.Import the Necessary Libraries

```
import numpy as np
import pandas as pd
import os
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

2. Reading the .csv dataset

```
data=pd.read_csv("../input/sms-spam-collection-dataset/spam.csv",encoding="latin")
data.head()
```

```
data.columns
```

3. Drop the unnamed Columns

```
data=data.drop(columns=["Unnamed: 2","Unnamed: 3","Unnamed: 4"])
```

4. Renaming Column names sensible

```
data=data.rename(
{
    "v1":"Category",
    "v2":"Message"
},
    axis=1
)
```

```
data.head()
```

5. Check for null values in dataset

```
data.isnull().sum()
```

```
data.info()
```

6.Creating a new Field to store the Message Lengths

```
data["Message Length"]=data["Message"].apply(len)
```

7. Histogram Inference of Message Lengths of Spam and Non-spam messages

```
fig=plt.figure(figsize=(12,8))
sns.histplot(
    x=data["Message Length"],
    hue=data["Category"]
)
plt.title("ham & spam messege length comparision")
plt.show()
```

```
ham_desc=data[data["Category"]=="ham"]["Message Length"].describe()
spam_desc=data[data["Category"]=="spam"]["Message Length"].describe()

print("Ham Messege Length Description:\n",ham_desc)
print("************************************")
print("Spam Message Length Description:\n",spam_desc)
```

```
data.describe(include="all")
```

8. Visualizing count of messages of Spam and Non Spam

```
data["Category"].value_counts()
```

```
sns.countplot(
    data=data,
    x="Category"
)
plt.title("ham vs spam")
plt.show()
```

```
ham_count=data["Category"].value_counts()[0]
spam_count=data["Category"].value_counts()[1]
```

```
total_count=data.shape[0]

print("Ham contains:{:.2f}% of total data.".format(ham_count/total_count*100))
print("Spam contains:{:.2f}% of total data.".format(spam_count/total_count*100))
```

9. Undersampling to Genralize Model and Balance Spam and Ham quantities in dataset

```
minority_len=len(data[data["Category"]=="spam"])
majority_len=len(data[data["Category"]=="ham"])
minority_indices=data[data["Category"]=="spam"].index
majority_indices=data[data["Category"]=="ham"].index
random_majority_indices=np.random.choice(
    majority_indices,
    size=minority_len,
    replace=False
)

undersampled_indices=np.concatenate([minority_indices,random_majority_indices])
df=data.loc[undersampled_indices]
df=df.sample(frac=1)
df=df.reset_index()
df=df.drop(
    columns=["index"],
)



df.shape


df["Category"].value_counts()


sns.countplot(
    data=df,
    x="Category"
)
plt.title("ham vs spam")
plt.show()
```

Display the head of new **df**

```
df.head()
```

10. Binary Encoding of Spam and Ham Categories

```
df["Label"]=df["Category"].map(
```

```
    {
        "ham":0,
        "spam":1
    }
)


df.head()
```

11. Import Necessary Libraries to perform Word Tokenization

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

stemmer=PorterStemmer()


corpus=[]
for message in df["Message"]:
    message=re.sub("[^a-zA-Z]"," ",message)
    message=message.lower()
    message=message.split()
    message=[stemmer.stem(words)
             for words in message
              if words not in set(stopwords.words("english"))
             ]
    message=" ".join(message)
    corpus.append(message)
```

12. Perform One Hot on Corpus

```
from tensorflow.keras.preprocessing.text import one_hot
vocab_size=10000

oneHot_doc=[one_hot(words,n=vocab_size)
            for words in corpus
            ]


df["Message Length"].describe()


fig=plt.figure(figsize=(12,8))
sns.kdeplot(
    x=df["Message Length"],
    hue=df["Category"]
)
```

```
plt.title("ham & spam messege length comparision")
plt.show()
```

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
sentence_len=200
embedded_doc=pad_sequences(
    oneHot_doc,
    maxlen=sentence_len,
    padding="pre"
)
```

```
extract_features=pd.DataFrame(
    data=embedded_doc
)
target=df["Label"]
```

```
df_final=pd.concat([extract_features,target],axis=1)
```

```
df_final.head()
```

13. Splitting Dependent and Independent Variables

```
X=df_final.drop("Label",axis=1)
y=df_final["Label"]
```

14. Train, test and Validation Split

```
from sklearn.model_selection import train_test_split
```

```
X_trainval,X_test,y_trainval,y_test=train_test_split(
    X,
    y,
    random_state=42,
    test_size=0.15
)
```

```
X_train,X_val,y_train,y_val=train_test_split(
    X_trainval,
    y_trainval,
    random_state=42,
```

```
    test_size=0.15
`
```

15.Building a Sequential Model

```
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Embedding
from tensorflow.keras.models import Sequential



model=Sequential()


feature_num=100
model.add(
    Embedding(
        input_dim=vocab_size,
        output_dim=feature_num,
        input_length=sentence_len
    )
)
model.add(
    LSTM(
    units=128
    )
)

model.add(
    Dense(
        units=1,
        activation="sigmoid"
    )
)
model.summary()


from tensorflow.keras.optimizers import Adam
model.compile(
    optimizer=Adam(
    learning_rate=0.001
    ),
    loss="binary_crossentropy",
    metrics=["accuracy"]
)
```

16. Model Fitting

```
history=model.fit(
    X_train,
```

```
    y_train,
    validation_data=(
        X_val,
        y_val
    ),
    epochs=10
)


metrics = pd.DataFrame(history.history)
metrics.rename(columns = {'loss': 'Training_Loss', 'accuracy': 'Training_Accuracy', 'val_]
def plot_graph_acc(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_graph_acc('Training_Accuracy', 'Validation_Accuracy', 'accuracy')


y_pred=model.predict(X_test)
y_pred=(y_pred>0.5)


model.save('Spam_SMS_classifier.h5')
```

17. Evaluating the Model

```
from sklearn.metrics import accuracy_score,confusion_matrix


score=accuracy_score(y_test,y_pred)
print("Test Score:{:.2f}%".format(score*100))


cm=confusion_matrix(y_test,y_pred)
fig=plt.figure(figsize=(12,8))
sns.heatmap(
    cm,
    annot=True,
)
plt.title("Confusion Matrix")
cm
```

18. Function to Test the Model on a Random message

```
def classify_message(model,message):
```

```
    for sentences in message:
        sentences=nltk.sent_tokenize(message)
        for sentence in sentences:
            words=re.sub("[^a-zA-Z]"," ",sentence)
            if words not in set(stopwords.words('english')):
                word=nltk.word_tokenize(words)
                word=" ".join(word)
    oneHot=[one_hot(word,n=vocab_size)]
    text=pad_sequences(oneHot,maxlen=sentence_len,padding="pre")
    predict=model.predict(text)
    if predict>0.5:
        print("It is a spam")
        print("predict score: ", predict[0][0])
    else:
        print("It is not a spam")
        print("predict score: ", predict[0][0])
```

```
message1="I am having my Tests right now. Will call back as soon as possible! Till then be
message2="Your Rs.8850 welcome bonus is ready to be credited. Download Junglee Rummy now.
```

```
classify_message(model,message1)
```

```
classify_message(model,message2)
```