**Assignment -4**

Python Programming

|  |  |
| --- | --- |
| Assignment Date | 21 October 2022 |
| Student Name | Shiny.R |
| Student Roll Number | 311419205036 |
| Maximum Marks | 2 Marks |

**Problem Statement :-** SMS SPAM Classification

Problem Statement: Over recent years, as the popularity of mobile phone devices has increased, Short Message Service (SMS) has grown into a multi-billion dollar industry. At the same time, reduction in the cost of messaging services has resulted in growth in unsolicited commercial advertisements (spams) being sent to mobile phones. Due to Spam SMS, Mobile service providers suffer from some sort of financial problems as well as it reduces calling time for users. Unfortunately, if the user accesses such Spam SMS they may face the problem of virus or malware. When SMS arrives at mobile it will disturb mobile user privacy and concentration. It may lead to frustration for the user. So Spam SMS is one of the major issues in the wireless communication world and it grows day by day

* Download the Dataset:- Dataset
* Import required library
* Read dataset and do pre-processing
* Create Model
* Add Layers (LSTM, Dense-(Hidden Layers), Output)
* Compile the Model
* Fit the Model
* Save The Model
* Test The Mode

**Solution:**

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

import re

import nltk

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

from tensorflow.keras.preprocessing.text import one\_hot

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.layers import LSTM

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Embedding

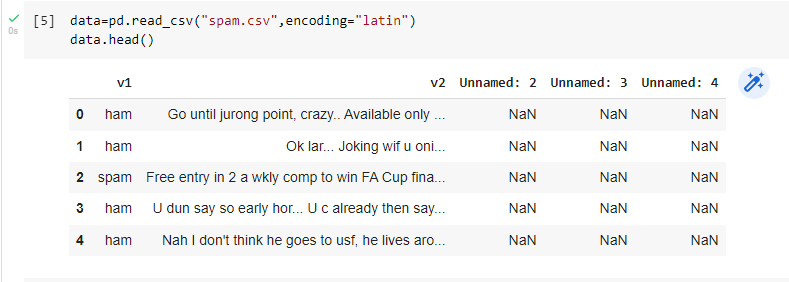
from tensorflow.keras.models import Sequential

from tensorflow.keras.optimizers import Adam

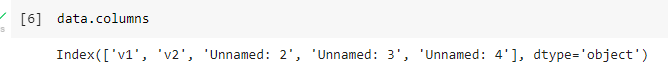
from sklearn.metrics import accuracy\_score,confusion\_matrix

data=pd.read\_csv("spam.csv",encoding="latin")

data.head()



data.columns



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

data=data.rename(

{

"v1":"Category",

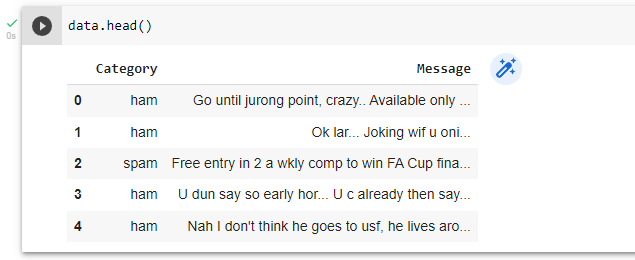
"v2":"Message"

},

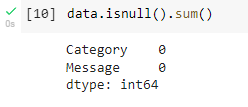
axis=1

)

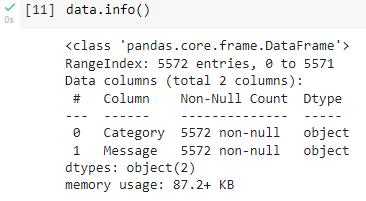
data.head()



data.isnull().sum()



data.info()



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

fig=plt.figure(figsize=(12,8))

sns.histplot(

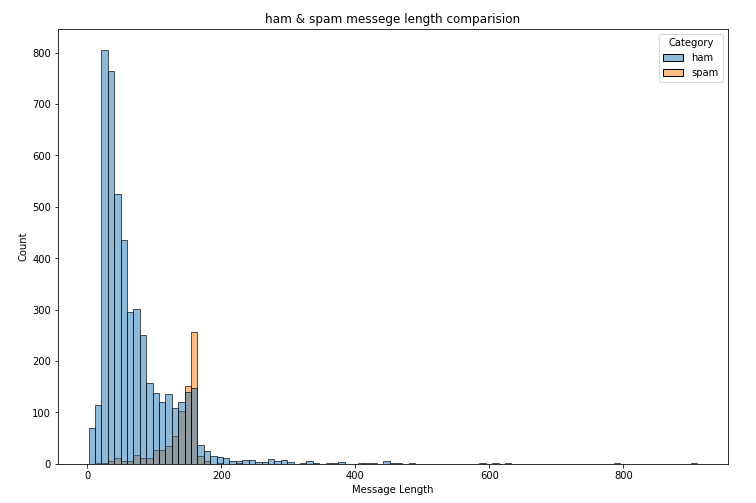
x=data["Message Length"],

hue=data["Category"]

)

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

plt.show()



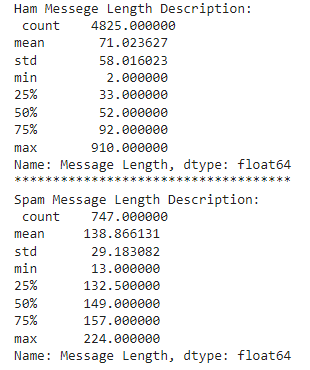
ham\_desc=data[data["Category"]=="ham"]["Message Length"].describe()

spam\_desc=data[data["Category"]=="spam"]["Message Length"].describe()

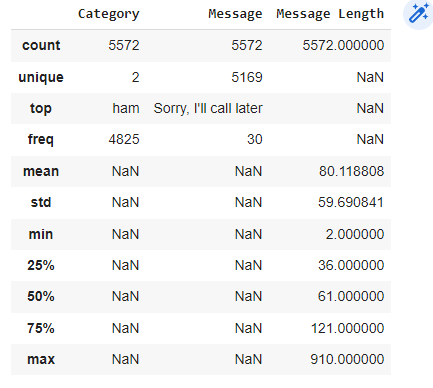
print("Ham Messege Length Description:\n",ham\_desc)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Spam Message Length Description:\n",spam\_desc)



data.describe(include="all")



data["Category"].value\_counts()



sns.countplot(

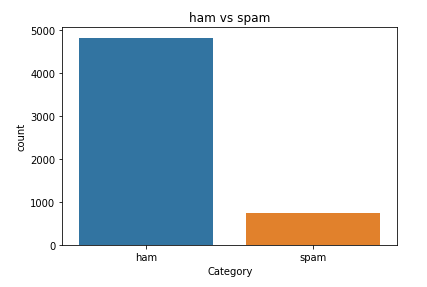
data=data,

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))



#compute the length of majority & minority class

minority\_len=len(data[data["Category"]=="spam"])

majority\_len=len(data[data["Category"]=="ham"])

#store the indices of majority and minority class

minority\_indices=data[data["Category"]=="spam"].index

majority\_indices=data[data["Category"]=="ham"].index

#generate new majority indices from the total majority\_indices

#with size equal to minority class length so we obtain equivalent number of indices length

random\_majority\_indices=np.random.choice(

majority\_indices,

size=minority\_len,

replace=False

)

#concatenate the two indices to obtain indices of new dataframe

undersampled\_indices=np.concatenate([minority\_indices,random\_majority\_indices])

#create df using new indices

df=data.loc[undersampled\_indices]

#shuffle the sample

df=df.sample(frac=1)

#reset the index as its all mixed

df=df.reset\_index()

#drop the older index

df=df.drop(

columns=["index"],

)

The resulting dataframes have \*\*1494\*\* rows and \*\*4\*\* columns

df.shape



df["Category"].value\_counts()



sns.countplot(

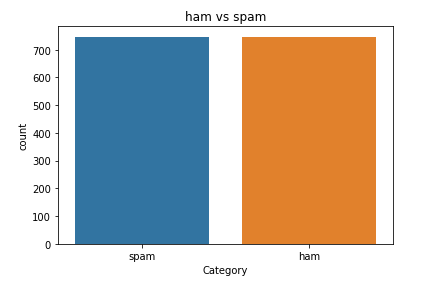
data=df,

x="Category"

)

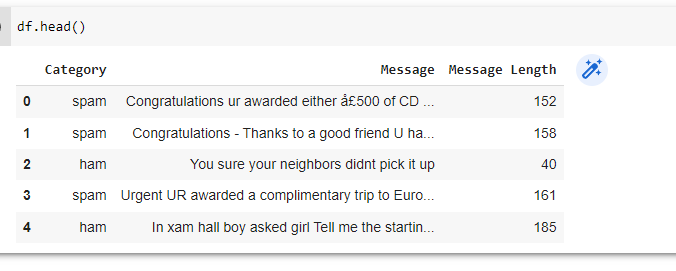
plt.title("ham vs spam")

plt.show()



Display the head of new \*\*df\*\*

df.head()



Created new column \*\*Label\*\* and encode \*\*ham\*\* as \*\*0\*\* and \*\*spam\*\* as \*\*1\*\*

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

{

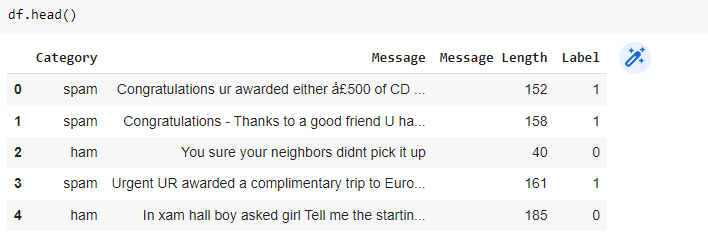
"ham":0,

"spam":1

}

)

df.head()



Import libraries to perform word \*\*tokenization\*\*

stemmer=PorterStemmer()

#declare empty list to store tokenized message

corpus=[]

#iterate through the df["Message"]

for message in df["Message"]:

#replace every special characters, numbers etc.. with whitespace of message

#It will help retain only letter/alphabets

message=re.sub("[^a-zA-Z]"," ",message)

#convert every letters to its lowercase

message=message.lower()

#split the word into individual word list

message=message.split()

#perform stemming using PorterStemmer for all non-english-stopwords

message=[stemmer.stem(words)

for words in message

if words not in set(stopwords.words("english"))

]

#join the word lists with the whitespace

message=" ".join(message)

#append the message in corpus list

corpus.append(message)

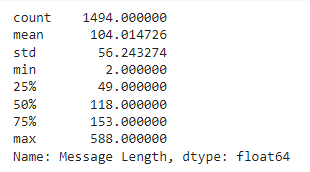
vocab\_size=10000

oneHot\_doc=[one\_hot(words,n=vocab\_size)

for words in corpus

]

df["Message Length"].describe()



fig=plt.figure(figsize=(12,8))

sns.kdeplot(

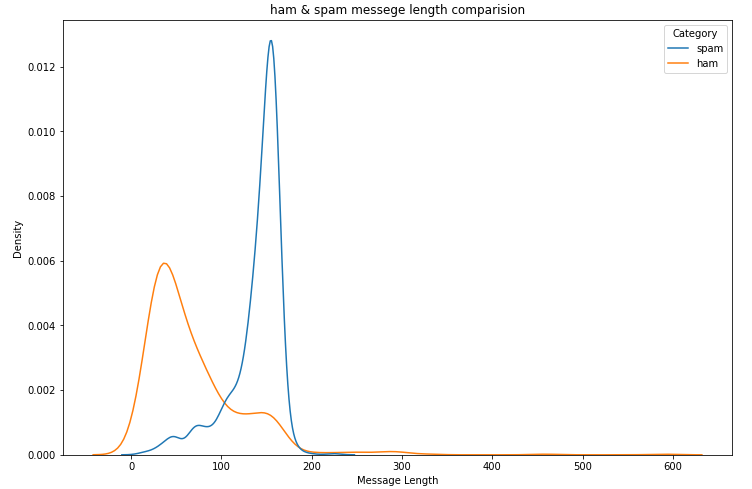
x=df["Message Length"],

hue=df["Category"]

)

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

plt.show()



sentence\_len=200

embedded\_doc=pad\_sequences(

oneHot\_doc,

maxlen=sentence\_len,

padding="pre"

)

extract\_features=pd.DataFrame(

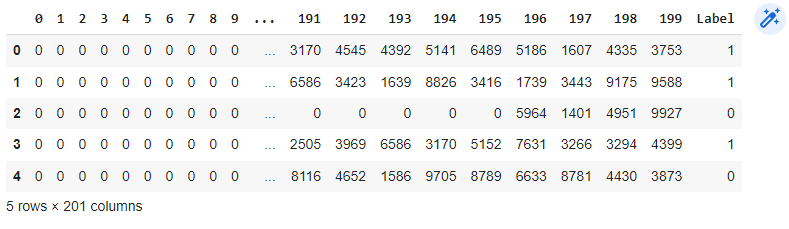
data=embedded\_doc

)

target=df["Label"]

df\_final=pd.concat([extract\_features,target],axis=1)

df\_final.head()



X=df\_final.drop("Label",axis=1)

y=df\_final["Label"]

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

)

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.compile(

optimizer=Adam(

learning\_rate=0.001

),

loss="binary\_crossentropy",

metrics=["accuracy"]

)

model.fit(

X\_train,

y\_train,

validation\_data=(

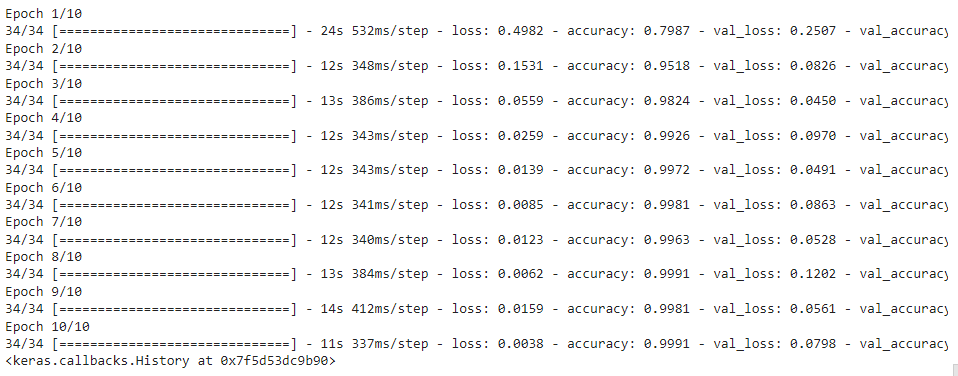
X\_val,

y\_val

),

epochs=10

)



y\_pred=model.predict(X\_test)

y\_pred=(y\_pred>0.5)



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(

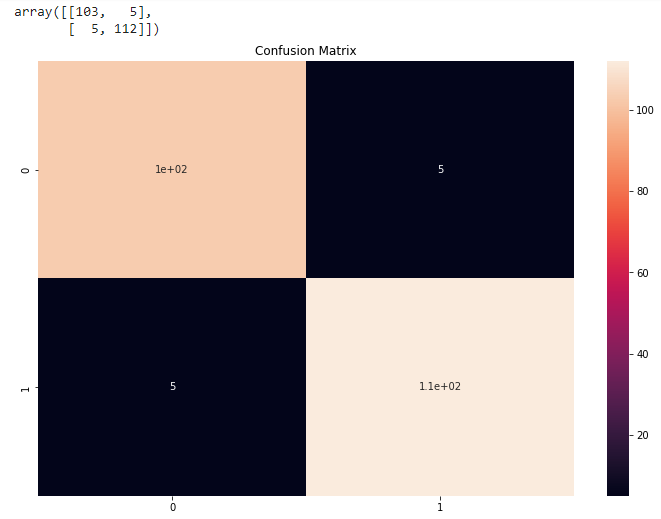
cm,

annot=True,

)

plt.title("Confusion Matrix")

cm



#The function take model and message as parameter

def classify\_message(model,message):

#We will treat message as a paragraphs containing multiple sentences(lines)

#we will extract individual lines

for sentences in message:

sentences=nltk.sent\_tokenize(message)

#Iterate over individual sentences

for sentence in sentences:

#replace all special characters

words=re.sub("[^a-zA-Z]"," ",sentence)

#perform word tokenization of all non-english-stopwords

if words not in set(stopwords.words('english')):

word=nltk.word\_tokenize(words)

word=" ".join(word)

#perform one\_hot on tokenized word

oneHot=[one\_hot(word,n=vocab\_size)]

#create an embedded documnet using pad\_sequences

#this can be fed to our model

text=pad\_sequences(oneHot,maxlen=sentence\_len,padding="pre")

#predict the text using model

predict=model.predict(text)

#if predict value is greater than 0.5 its a spam

if predict>0.5:

print("It is a spam")

#else the message is not a spam

else:

print("It is not a spam")

message1="I am having a bad day and I would like to have a break today"

message2="This is to inform you had won a lottery and the subscription will end in a week so call us."

classify\_message(model,message1)



classify\_message(model,message2)

