Assignment-4 (SMS SPAM Classification)

import numpy as np import. pandas.as.pd import.seaborn.as.sns import matplotlib.pyplot as plt

data = pd.read_csv('/content/sample_data/spam.csv', delimiter=', ', encoding='latin-1') **data.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		Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3					

```
spam 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

```
data.columns
```

```
Index(['vi', 'V2', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'],
dtype='object')
```

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

```
#rename the two relevant columns data-data.rename(
{
"V1":"Category",
"V2":"Message" }, axis=1)
```

```
data.head()
```



```
Category
```

Message

Go until jurong point, crazy.. Available only ...

ham **ham**

1

Ok lar... Joking wif u oni...

spam

Free entry in 2 a wkly comp to win FA Cup fina...

ham

U dun say so early hor... U c already then say...

ham Nah I don't think he goes to usf. he lives aro... `#check for null values`

`data.isnull().sum()`

Category 0 Message dtype: int64

`data.info()`

<class 'pandas.core. frame. DataFrame'> Range Index: 5572 entries, 0 to 5571 Data columns (total 2 columns):

Column Non-Null Count Dtype --- ------ -------------- -----

`0 Category 5572 non-null object`

`1 Message 5572 non-null object` **dtypes: object(2) memory usage: 87.2+ KB**

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

fig=`plt.figure(figsize=(12,8))` sns.histplote

X=data["Message Length"], hue=`data["Category"]`

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

ham & spam messege length comparision

Category O ham
spam
O
Count

300 +

#Display the description of length of ham and spam messages seperately on an individual serie

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("Spam Message Length Description:\n", spam_desc)

min
Ham Messege Length Description:
count 4825.000000 mean
71.023627 std
58.016023
2.000000 25%
33.000000
52.000000 75%
92.000000 max
910.000000 Name: Message Length, dtype: float64
50%

****

Spam Message Length Description:
count 747.000000 mean 138.866131 std
29.183082 min
13.000000 25% 132.500000 50% 149.000000 75% 157.000000 max
224.000000 Name: Message Length, dtype: float64

data.describe(include="all")

	Category	Message	Message Length
count	5572	5572	5572.000000
unique	5169		
top	ham	Sorry, I'll call later	
freq	4825	30	
mean	NaN	NaN	80.118808

std
NaN
NaN
59.690841

min
NaN
NaN
2.000000 36.000000
25%
NaN
NaN

50%
NaN
NaN
61.000000

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

ham 4825 spam 747 Name: Category, dtype: int64

```
sns.countplot(
data=data, x="Category"

plt.title("ham vs spam") plt.show()
```

ham vs spam
50001

4000

3000
count
2000

1000

ham
spam

Category

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

total_count=data.shape [0]
```

https://colab.research.google.com/drive/1RUT5WjmG_-tjTK299qMNKS96wYXYODM
m#scrollTo=4762FS4CDqL&printMode=true


```
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))
```

Ham contains:86.59% of total data. Spam contains:13.41% of total data.

```
#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 dfsdf.reset_index()

#drop the older index df=df.drop
columns=["index"],

df.shape

(1494, 3)

df["Category"].value_counts()

ham 747 spam 747 Name: Category, dtype: int64

sns.countplot(
data-df, x="Category"

plt.title("ham vs spam") plt.show()

ham vs spam
count

ham
spam
Category

```
df.head()
```

Category
Message Message Length

ham
Aah! A cuddle would be lush! I'd need lots of ...

ham
I'm in solihull, do you want anything?

spam
Double Mins & 1000 txts on Orange tariffs. Lat...

ham

spam
No we put party 7 days a week and study light...

URGENT!! Your 4* Costa Del Sol Holiday or å£50...

```
#Created new column Label and encode ham as 0 and spam as 1
df["Label"]=df["Category"].map

"ham":0, "spam":1
```

df.head()

Category
Message Message Length Label

ham
Aah! A cuddle would be lush! I'd need lots of ...
87
0

1.
ham
I'm in solihull, do you want anything?
40
0

spam
Double Mins & 1000 txts on Orange tariffs. Lat...
151
1

ham
No we put party 7 days a week and study lightl...
1260

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

```
stemmer=PorterStemmer()


nltk.download('stopwords')

[nltk_data] Downloading package stopwords to /root/nltk_data... [nltk_data]
Unzipping corpora/stopwords.zip.  True


#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)
```



```
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()
```

```
count 1494.000000 mean 105.203481
std
61.166448 min
3.000000 25%
48.000000 50% 118.000000 75% 153.000000 max 790.000000 Name: Message
Length, dtype: float64
```

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

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

ham & spam messege length comparision

Category

```
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.Data
Frame
    data=embedded_
    doc
```

```
target=df["label"]
```

W.Utili

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

0.0

02

+

V

```
df_final.head()
```

	1	2	3	4	5	6	7	8	9	...	191	192	193	194	195	196	197	198	1
	0	0	0	0	0	0	0	0	0	...	20	90	1632	42	89	7158	478	5	808
	6	13	3	8	3	4	8	41	1	0	0	0	0	0	0	0	...	0	0
	2	0	0	0	0	0	0	0	0	...	12	75	702	1694	4	114	41	62	39
	8	5	3	6	7	2	3	0	0	0	0	0	0	0	0	0	0	37	05
	9	8	8	3	4	5	0	0	8	0	3	0	8	6	3	0	29	4	0

5 rows x 201 columns

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

```
from sklearn.model_selection import
train_test_split

X_trainval,x_test, y_trainval,y_test=train_te
st_split(
    X
    ,
    y,
    random_state=4
    2,
    test_size=0.15


X_train,x_val,y_train, y_val=train_tes
t_split(
    X_trainval
    ,
```



```
    y_trainval, random_state=42, test_size=0.15



    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(
LSTMC units=128

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

from tensorflow.keras.optimizers import Adam model.compiled
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

Epoch 1/10 34/34 [==============================] - 8s 33ms/step - loss: 0.5258 - accuracy: 0.7653 Epoch 2/10 34/34 [==============================] - 1$ 16ms/step - loss: 0.1718 -

accuracy: 0.9453 Epoch 3/10 34/34
[==============================] - 1s 16ms/step - loss: 0.0533 - accuracy: 0.9842 Epoch 4/10 34/34
[==============================] - 1s 15ms/step - loss: 0.0254 - accuracy: 0.9926 Epoch 5/10 34/34
[==============================] - 1s 16ms/step - loss: 0.0184 - accuracy: 0.9954 Epoch 6/10 34/34
[==============================] - 1s 16ms/step - loss: 0.0134 - accuracy: 0.9963 Epoch 7/10 34/34
[==============================] - 1s 16ms/step - loss: 0.0150 - accuracy: 0.9954 Epoch 8/10 34/34
[==============================] - 1s 16ms/step - loss: 0.0112 - accuracy: 0.9972 Epoch 9/10 34/34
[==============================] - 1s 16ms/step - loss: 0.0062 - accuracy: 0.9981 Epoch 10/10 34/34
[==============================] - 1s 16ms/step - loss: 0.0050 - accuracy: 0.9991 <keras.callbacks.History at 0x7fa3263a7850>

=
=
s
accuracy

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

=
==
==
===
=====
=====
=
=====
=

```
from sklearn.metrics import accuracy_score, confusion_matrix

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

```
Test Score:96.00%
```

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

plt.title("Confusion Matrix")
cm
```

array([[100, 2],
[7, 116]])

Confusion Matrix

- 100

le+02

-40

12e+02

#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

nitk.download('punkt')

[nltk_data] Downloading package punkt to /root/nltk_data... [nltk_data] Unzipping tokenizers/punkt.zip. True

classify_message(model, message1)

1/1 [==============================] - 0s 21ms/step It is not a spam

classify_message(model, message2)

1/1 [==============================] - 0s 22ms/step It is a spam

Colab paid products - Cancel contracts here