Assignment-4 (SMS SPAM Classification)

import numpy as np import. p**andas.a**s.pd import.**seaborn.as.sns** import matplotlib.pyplot as plt

data = pd.read\_cs*v('/c*ontent/sample\_data*/*spam.cs*v*', delimiter=', ', encoding='latin-1') **data.head()**

**v1**

***V*2 Unnamed: 2 Unnamed: 3 Unnamed: 4**

0

ham

NaN

NaN

NaN

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

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

1

ham

NaN

NaN

NaN

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

NaN

NaN

NaN

2 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"])

**#renam**e the two relevant columns **data-data.rename(**

{

"V1":"Category",

"V2":"Message" **}, axis=1)**

data.head()

https*://*c**olab.resear**ch.google.com*/*dri*v*e*/*1RUT5WjmG\_-tjTK299qMNK**S96wYxYODMm#scrollTo=47k2FS4CDrqL&printMode=true**

***1/1*4**

1*0/2*9*/*22, 11:33 AM

**Assignme**nt\_4.ipynb - Cola**boratory**

**Categor*y***

**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.cor**e. frame. DataFrame'> Range Index: 5572 entries, 0 to 5571 **Data col**umns (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=da**ta["Message Length"], hue=data["Category"]

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

https*://*c**olab.resear**ch.google.com*/*dri*v*e*/*1RUT5WjmG\_-tjTK299qMNK**S96wYxYODMm#scrollTo=47k2FS4CDrqL&printMode=true**

2*/1*4

*1*0*/29/*22, 11:33 AM

**Assignment\_4.ipynb - Colaboratory**

ham & spam messege length comparision

Category O ham

spam

O

Count

300 +

#Display the description of length of ham a**nd spam messages seperatel**y on an individual serie

**IA**

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 L**ength 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 L**ength, 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**

**dat**a.describe(include="all")

https*://*c**olab.resear**ch.google.com*/*dri*v*e*/*1RUT5WjmG\_-tjTK299qMNK**S96wYxYODMm#scrollTo=47k2FS4CDrqL&printMode=true**

3/14

1*0/2*9*/*22, 11:33 AM

**Assig**nment\_4.jpynb - Col**aboratory**

**Category**

**Message Message Length**

**count**

**5572**

**5572**

5572.000000

**unique**

**5169**

NaN

**top**

ham

Sorry, I'll call later

NaN

**freq**

**4825**

30

NaN

**mean**

NaN

NaN

80.118808

**st*d***

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 7*47* Name: Categ**ory, 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.researc**h.google.com*/d*ri*ve/*1RUT5WjmG\_-tjTK299qMNKS96wYXYODM**m#scrollTo=4762FS4CDqL&printMode=true**

***4/*14**

10*/*29/22, 11:33 AM

**Assignment\_4.ipynb - Colaboratory** 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 to**tal data. Spam contains:13.41% of total data.**

#compute the length of majority & minority c**lass** minority\_len=len (data[data["Category"]=="spam"]) majority\_len=len(data[data["Category"]=="ham"])

**#st**ore the indices of majority and minority c**lass** 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()

**#d**rop the older index df=df.drop

columns=["index"],

df.shape

(1494, 3)

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

**ham 747 spam 7*47*** Name: Category, dtype: int64

https*://c*ola**b.researc**h.google.com*/*drive*/*1RUT5WjmG\_-tjTK299qMNKS96w*Y*xYODMm#scrollTo=47k2FS4CDrqL&printMo**de=true**

***5/14***

1*0/2*9*/*22, 11:33 AM

**Assignment**\_4.ipynb - C**olaboratory**

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 col**umn Label and encode ham as O and spam as 1 df["Label"]=df["Category"].map

"ham":0, "spam":1

df.head()

https*://*c**olab.resear**ch.google.com*/*dri*v*e*/*1RUT5WjmG\_-tjTK299qMNK**S96wYxYODMm#scrollTo=47k2FS4CDrqL&printMode=true**

**6*/*14**

1*0/2*9*/*22, 11:33 AM

**Assig**nment\_4.ipynb - Col**aboratory**

**Categor*y***

**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 impo**rt 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=[]

**#iterat**e through the df["Message"] for message in df["Message"]:

**#replace e*v*ery special characte**rs, numbers etc.. with whitespace of message #It will help retain only letter*/*alphabets **messa**ge=re. sub("[^a-zA-Z]", " ", message)

**#convert every letters to its lowercase message=message. lower()**

#split the word into individual word list **message=message.s**plit()

#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 th**e whitespace message**=" ".join(message)

#append the message in corpus list corpus.append(message)

https*://*c**olab.resear**ch.google.com*/*dri*v*e*/*1RUT5WjmG\_-tjTK299qMNK**S96wYxYODMm#scrollTo=47k2FS4CDrqL&printMode=true**

***7/14***

10*/*29/22, 11:33 AM

**Assignment\_4.ipynb - Colaboratory** from tensorflow.ker**as.preprocessing. text** import one\_hot **vocab\_size=10000**

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

**for words i**n 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: Messa**ge 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()

https*://*c**olab.resear**ch.google.com*/*dri*v*e*/*1RUT5WjmG\_-tjTK299qMNK**S96wYxYODMm#scrollTo=47k2FS4CDrqL&printMode=true**

**8*/*14**

1*0/2*9*/*22, 11:33 AM

**Assignment\_4.ipynb - Colaboratory**

ham & spam messege length comparision

Category

from tensorflo**w.keras.preprocessing. sequence import pad\_sequences sentence\_len=200** embedded\_doc=pad\_sequences (

oneHot\_doc, maxlen=sentence\_len, padding="pre"

**extr**act\_features=pd.DataFrame

data=embedded\_doc

target=df["label"]

W.Ut*i*li

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

0.002 +

V df\_final.head()

**1 2 3 4 5 6 7 8** 9 ... **191 192 193 194 195 196 19*7* 198 1** 00000000000 ... 2090 1632 42897158 *47*8 5808 61338348 41 10000000000... 0 0 0 0 0 0 86634425 66 20000000000 ... 1*275* 702 1694 4114 4162 3935 4162 8536 *7*2 3 0 0 0 0 0 0 0 0 0 0 3705 9946 5462 7158 98834500 8030 8630 29 4 0000000000 ... *47*536414 5018 1953 216 11*7*5 8861 2485 60

**5 rows x 201 columns**

X=df\_final.drop ("Label", a**xis=1)** y=df\_final["label"]

from sklearn.model\_selection import train\_**test\_split**

X\_trainval,x\_test,y\_trainval,y\_test=train\_test\_split(

**X,**

**y,**

random\_st**ate=42, test\_size=0.15**

X\_train,x\_val,y\_train,y\_val=train\_test\_split(

X\_trainval,

https*://c*ola**b.researc**h.google.com*/*drive*/*1RUT5WjmG\_-tjTK299qMNKS96w*Y*xYODMm#scrollTo=47k2FS4CDrqL&printMo**de=true**

**9*/1*4**

**Assignment**\_4.ipynb - C**olaboratory**

*1*0*/29/*22, 11:33 AM

y\_trainval, random\_state=42, **test\_size=0.15**

from tensorflow.ke**ras.layers imp**ort LSTM from tensorflow.ke**ras.layers import Dense** from tensorflow.ke**ras.layer**s 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=s**entence\_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**

https*://*c**olab.resear**ch.google.com*/*dri*v*e*/*1RUT5WjmG\_-tjTK299qMNK**S96wYxYODMm#scrollTo=47k2FS4CDrqL&printMode=true**

10*/*14

**Assignme**nt\_4.ipynb - Cola**boratory**

1*0/2*9*/*22, 11:33 AM

**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: 0.17**18 - 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: 0.99*7*2 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, confusi**on\_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**

https*://*c**olab.resear**ch.google.com*/*dri*v*e*/*1RUT5WjmG\_-tjTK299qMNK**S96wYxYODMm#scrollTo=47k2FS4CDrqL&printMode=true**

**1*1/1*4**

1*0/2*9*/*22, 11:33 AM

**Assignme**nt\_4.ipynb - Cola**boratory**

array([[100, 2],

[ 7, 116]])

Confusion Matrix

- 100

le+02

-40

12e+O2

#The function take model and m**essage as parameter def classify\_message(model, message):**

**#We will treat message as a para**graphs containing multiple sentences(lines) **#we will ext**ract 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 d**ocumnet using pad**\_sequences** #this can be fed to our model text=pad\_sequences (oneHot,maxlen=sentence\_len, padding="pre")

#predict the text using model

https*://c*ola**b.researc**h.google.com*/*drive*/*1RUT5WjmG\_-tjTK299qMNKS96w*Y*xYODMm#scrollTo=47k2FS4CDrqL&printMo**de=true**

**12*/*14**

**Assignment\_4.ipynb - Colaboratory**

1*0/2*9*/*22, 11:33 AM

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")

**message**1="I am having a bad day and I would like to have a break today" **messa**ge2="This is to inform you had won a lottery and the subscription will e**nd 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 [==============================**] - Os 21ms/step It is not a spam

**classify\_message(**model, message2)

**1/1 [==============================**] - Os 22ms*/*step It is a spam

https*://*c**olab.resear**ch.google.com*/*dri*v*e*/*1RUT5WjmG\_-tjTK299qMNK**S96wYxYODMm#scrollTo=47k2FS4CDrqL&printMode=true**

13*/*14

1*0/2*9*/*22, 11:33 AM

**Assignment\_4.ipynb - Colaboratory**

Colab paid products - Cancel contracts here

https*://*c**olab.resear**ch.google.com*/*dri*v*e*/*1RUT5WjmG\_-tjTK299qMNK**S96wYxYODMm#scrollTo=47k2FS4CDrqL&printMode=true**

***1*4*/*14**