# This Python 3 environment comes with many helpful analytics libraries installed

# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python

# For example, here's several helpful packages to load

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

# Input data files are available in the read-only "../input/" directory

# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os

for dirname, \_, filenames in os.walk('/kaggle/input'):

for filename in filenames:

print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"

# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session

import tensorflow\_hub as hub

import tensorflow\_text as text

idf=pd.read\_csv('/kaggle/input/sms-spam-collection-dataset/spam.csv', encoding = "ISO-8859-1")

df.head(5)mportdf.df.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'],inplace=True)

df.rename(columns = {'v1':'Category', 'v2':'Message'}, inplace = True)

df.head(5)columns tensordf.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 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+ KBflow as tfdf.groupby('Category').describe()print('% Imbalanced Data:',747/4825)

df\_spam = df[df['Category']=='spam']

df\_spam.shape

df\_ham = df[df['Category']=='ham']

df\_ham.shape

df\_ham\_downsampled = df\_ham.sample(df\_spam.shape[0])

df\_ham\_downsampled.shape

df\_balanced = pd.concat([df\_ham\_downsampled, df\_spam])

df\_balanced.shapedf\_balanced = pd.concat([df\_ham\_downsampled, df\_spam])

df\_balanced.shape

df\_balanced.shapedf\_balanced['spam']=df\_balanced['Category'].apply(lambda x: 1 if x=='spam' else 0)

df\_balanced.sample(5)df\_balanced['spam']=df\_balanced['Category'].apply(lambda x: 1 if x=='spam' else 0)

df['spam']=df['Category'].apply(lambda x: 1 if x=='spam' else 0)

df.head()from sklearn.model\_selection import train\_test\_split

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

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

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

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

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

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

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Message'],df\_balanced['spam'], stratify=df\_balanced['spam'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Message'],df\_balanced['spam'], stratify=df\_balanced['spam'])

X\_train.head()

bert\_preprocess = hub.KerasLayer("https://tfhub.dev/tensorflow/bert\_en\_uncased\_preprocess/3")

bert\_encoder = hub.KerasLayer("https://tfhub.dev/tensorflow/bert\_en\_uncased\_L-12\_H-768\_A-12/4")

def get\_sentence\_embeding(sentences):

preprocessed\_text = bert\_preprocess(sentences)

return bert\_encoder(preprocessed\_text)['pooled\_outpu

c=get\_sentence\_embeding(['mango','banana','Narendra modi','data analytics','natural language processing'])

c=get\_sentence\_embeding(['mango','banana','Narendra modi','data analytics','natural language processing'])

from sklearn.metrics.pairwise import cosine\_similarity

cosine\_similarity([c[0]],[c[1]])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Message'],df\_balanced['spam'], stratify=df\_balanced['spam'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Message'],df\_balanced['spam'], stratify=df\_balanced['spam'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Message'],df\_balanced['spam'], stratify=df\_balanced['spam'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Message'],df\_balanced['spam'], stratify=df\_balanced['spam'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Message'],df\_balanced['spam'], stratify=df\_balanced['spam'])from sklearn.model\_selection import train\_test\_split

from sklearn.metrics.pairwise import cosine\_similarity

cosine\_similarity([c[0]],[c[1]])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Message'],df\_balanced['spam'], stratify=df\_balanced['spam'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Message'],df\_balanced['spam'], stratify=df\_balanced['spam'])from sklearn.model\_selection import train\_test\_split

from sklearn.metrics.pairwise import cosine\_similarity

cosine\_similarity([c[0]],[c[1]])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Message'],df\_balanced['spam'], stratify=df\_balanced['spam

cosine\_similarity([c[3]],[c[4]])

cosine\_similarity([c[1]],[c[2]])

# Bert layers

#Text input passed to bert\_preprocess

text\_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')

preprocessed\_text = bert\_preprocess(text\_input)

#Processed text passed to bert\_encoder

outputs = bert\_encoder(preprocessed\_text)

# Neural network layers

#Dropout

l = tf.keras.layers.Dropout(0.1, name="dropout")(outputs['pooled\_output'])

l = tf.keras.layers.Dense(1, activation='sigmoid', name="output")(l)

# Use inputs and outputs to construct a final model

model = tf.keras.Model(inputs=[text\_input], outputs = [l])

model.summary()

len(X\_train)

METRICS = [tf.keras.metrics.BinaryAccuracy(name='accuracy'),tf.keras.metrics.Precision(name='precision'),

tf.keras.metrics.Recall(name='recall')]

model.compile(optimizer='adam',loss='binary\_crossentropy',metrics=METRICS)

odel.fit(X\_train, y\_train, epochs=10)

Epoch 1/10

35/35 [==============================] - 16s 215ms/step - loss: 0.6320 - accuracy: 0.6616 - precision: 0.6654 - recall: 0.6500

Epoch 2/10

35/35 [==============================] - 8s 220ms/step - loss: 0.5041 - accuracy: 0.8134 - precision: 0.7980 - recall: 0.8393

Epoch 3/10

35/35 [==============================] - 8s 216ms/step - loss: 0.4292 - accuracy: 0.8625 - precision: 0.8524 - recall: 0.8768

Epoch 4/10

35/35 [==============================] - 8s 216ms/step - loss: 0.3820 - accuracy: 0.8857 - precision: 0.8699 - recall: 0.9071

Epoch 5/10

35/35 [==============================] - 7s 212ms/step - loss: 0.3467 - accuracy: 0.8813 - precision: 0.8675 - recall: 0.

model.evaluate(X\_test, y\_test)

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

y\_predicted = y\_predicted.flatten()

12/12 [==============================] - 3s 207ms/step

#If probablity greater than 0.5 than make it 1 otherwise

from sklearn.metrics import confusion\_matrix, classification\_report

cm = confusion\_matrix(y\_test, y\_predicted)

cm

from matplotlib import pyplot as plt

import seaborn as sn

sn.heatmap(cm, annot=True, fmt='d')

plt.xlabel('Predicted')

plt.ylabel('Truth')

print(classification\_report(y\_test, y\_predicted))

precision recall f1-score support

0 0.91 0.90 0.91 187

1 0.90 0.91 0.91 187

accuracy 0.91 374

macro avg 0.91 0.91 0.91 374

weighted avg 0.91 0.91 0.91 374

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Message'],df\_balanced['spam'], stratify=df\_balanced['spam'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Message'],df\_balanced['spam'], stratify=df\_balanced[

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Message'],df\_balanced['spam'], stratify=df\_balanced['spam'])

reviews = [

'You are awarded a SiPix Digital Camera! call 09061221061 from landline. Delivery within 28days. T Cs Box177. M221BP. 2yr warranty. 150ppm. 16 . p pÂ£3.99',

'it to 80488. Your 500 free text messages are valid until 31 December 2005.',

'Reply to win Â£100 weekly! Where will the 2006 FIFA World Cup be held? Send STOP to 87239 to end service',

'Hey Sam, Are you coming for a cricket game tomorrow',

"Why don't you wait 'til at least wednesday to see if you get your ."]

model.predict(reviews)