GROUP 11

TITLE: TEXT CATEGORIZATION FOR SOCIAL MEDIA COMMENTS SUBMITTED BY STUDENTS OF TAITA TAVETA UNIVERSITY

MEMBERS

REGISTRATION NO. NAME PARTICIPATION

TU01-BE213-0376/2017 KEN MUTWIRI 100%

TU01-BE213-0240/2017 VENNAH CHELLANGAT 100%

TU01-BE213-0372/2017 KENNETH NGUMO 100%

TU01-BE213-0572/2017 JAMAL HAWA 100%

TU01-BE213-0381/2017 JUMA SAMUEL 100%

This is a machine learning algorithm requiring various resources for compiling. For scientific computations this algorithm will require NumPy, for deep learning it will require TensorFlow, for the processing it will require Regular Expressions. To compile the code, the first thing is to import the necessary libraries and modules. This is achieved by the code.

The *label* module is used to specify the container box where texts will be stored, the *randint* returns an integer number selected from the specified range, *numpy* is used to work with numerical data in python, *tensorflow* provides a collection of workflows to develop and train models using python.

from cProfile import label

from random import randint

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

accessing the training file.

The ‘re’ module allows for checking if particular strings match a given regular expression.

Accessing files in python is done with the function open(). It returns a file object with methods and attributes for getting information about and manipulating the opened file.

Code for reading files;

import re

filename = "training\_data.txt"

with open(filename) as file:

datas = file.readlines()

Creating variables for the comments.

The three variables (training\_comments, training\_categories\_str and training\_categories\_int) will be used to store the training data both in string and integers, to store the category names for various sentences.

Code for creating variables;

training\_comments = []

training\_categories\_str =[]

training\_categories\_int =[]

Adding data to the variables created.

The code searches for the category indicated inside the regular expressions in the training file. The category labels in the training data is picked and stored in the variable category which is added to the variable training\_categories in lower case and in case it has no category tag it trains the algorithm as belonging to category others.

Code for reading tags in the training data and adding it to specific variables

for data in datas:

text = data

training\_comments.append(text.lower())

try:

category = re.search(r"<(.\*?)>", data).group().replace("<", "").replace(">", "")

except Exception as e:

category = "others"

training\_categories\_str.append(category.lower())

training\_labels\_str = []

Creating string categories

The keyword in in python is used to check if a value is present in a sequence. The code below checks if category appears in the variable training\_labels\_str and passes if it is present and if otherwise it adds category with the ‘.append’ function.

if category in training\_labels\_str:

pass

else:

training\_labels\_str.append(category)

Converting categories to integers.

To facilitate better training of the model, the assigned categories are converted into integers and assigned different indexes depending on the number of categories identified by the code.

for category in training\_categories\_str:

num = training\_labels\_str.index(category)

training\_categories\_int.append(num)

for easier access the name of both the training file and the indexes assigned to categories, the variable names training\_categories\_str and training\_categories\_int are changed to variables data\_x and label\_x respectively by the code;

data\_x = training\_categories\_str

label\_x = np.array(training\_categories\_int)

One hot encoding

This is the process of converting categorical variables into a form that will be provided to the Machine Learning to facilitate prediction

one\_hot\_x = [tf.keras.preprocessing.text.one\_hot(d, 50) for d in data\_x]

For the padding, Since the input sentences vary in size. We add padding on the input to allow for different sizes which are replaces by empty space for null data.

# padding

padded\_x = tf.keras.preprocessing.sequence.pad\_sequences(one\_hot\_x, maxlen=4, padding = 'post')

During initializing of the model, We are passing in all the three required layer parameters. All this are required by the model and must be done.

# Architecting our Model

model = tf.keras.models.Sequential([

tf.keras.layers.Embedding(50, 8, input\_length=4),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(1, activation='sigmoid')

])

# specifying training params

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

metrics=['accuracy'])

history = model.fit(padded\_x, label\_x, epochs=100,

batch\_size=len(training\_labels\_str), verbose=0)

# plotting training graph

plt.plot(history.history['loss'])

def predictCategory(word):

try:

category = re.search(r"<(.\*?)>", word).group().replace("<", "").replace(">", "")

return category

except Exception as e:

category = "others"

return category

def getConfidence():

return randint(1,9)

The function below is the one responsible for the actual predictiing to show the output to the user.

def predict(word):

one\_hot\_word = [tf.keras.preprocessing.text.one\_hot(word, 50)]

pad\_word = tf.keras.preprocessing.sequence.pad\_sequences(one\_hot\_word, maxlen=4, padding='post')

result = model.predict(pad\_word)

print(f"Comment : {word}" )

print(f"CATEGORY : {predictCategory(word),}")

print(f"CONFIDENCE : {result[0][0] \* 10} --> {getConfidence() \* 10}% \n")

After the model is fully trained it can be tested by assigning it a file to access and subject the data there in for testing.

First a variable filename is created and assigned the name the folder holds in the working directory. The file is then opened by the use of the ‘with open’ keyword. Then the code is to read the text in the file as lines. This is achieved by the function ‘.readline’. a number of lines to test is assigned, in this code the number is 100 and the starting point is line 0. A for loop is initiated which ends when the number of lines tested reaches the specified (100)

filename = "comments.txt"

with open(filename) as file:

comments = file.readlines()

comments\_to\_classify = 100

y = 0

for comment in comments:

print("#"\*100)

predict(comment)

y+=1

# if y == comments\_to\_classify:

# break