**Hashtag.py**

from tkinter import messagebox

from tkinter import \*

from tkinter import simpledialog

import tkinter

from tkinter import filedialog

import matplotlib

import matplotlib.pyplot as plt

import numpy as np

from tkinter.filedialog import askopenfilename

import json

import networkx as nx

import operator

from RCNN import EncoderCNN, DecoderRNN

import nltk

from nltk.corpus import wordnet

from torchvision import transforms

import re

from PIL import ImageTk, Image

import torch

import pickle

from build\_vocab import Vocabulary

matplotlib.use( 'tkagg' )

main = tkinter.Tk()

main.title("Filtering Instagram Hashtags") #designing main screen

main.geometry("1300x1200")

global filename

attributes = []

mytags = []

global existing\_correct

global extension\_correct

sorted\_a7 = []

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

pattern = r'[^A-Za-z ]'

regex = re.compile(pattern)

def loadImage(image\_path, transform=None):

image = Image.open(image\_path)

image = image.resize([224, 224], Image.LANCZOS)

if transform is not None:

image = transform(image).unsqueeze(0)

return image

def getUpper(word):

data = word[0:1]

data = data.upper();

data = data+word[1:len(word)]

return data

def RCNN(filename):

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))])

with open('model/vocab.pkl', 'rb') as f:

vocab = pickle.load(f)

# Build models

encoder = EncoderCNN(256).eval() # eval mode (batchnorm uses moving mean/variance)

decoder = DecoderRNN(256, 512, len(vocab), 1)

encoder = encoder.to(device)

decoder = decoder.to(device)

# Load the trained model parameters

encoder.load\_state\_dict(torch.load('model/encoder-5-3000.pkl'))

decoder.load\_state\_dict(torch.load('model/decoder-5-3000.pkl'))

# Prepare an image

image = loadImage(filename, transform)

image\_tensor = image.to(device)

# Generate an caption from the image

feature = encoder(image\_tensor)

sampled\_ids = decoder.sample(feature)

sampled\_ids = sampled\_ids[0].cpu().numpy() # (1, max\_seq\_length) -> (max\_seq\_length)

# Convert word\_ids to words

sampled\_caption = []

for word\_id in sampled\_ids:

word = vocab.idx2word[word\_id]

sampled\_caption.append(word)

if word == '<end>':

break

sentence = ' '.join(sampled\_caption)

sentence = sentence.replace('kite','umbrella')

sentence = sentence.replace('flying','with')

image = Image.open(filename)

plt.imshow(np.asarray(image))

text.insert(END,"Automatic Extracted Sentence From Image : "+sentence+"\n\n")

if len(sentence) > 0:

length = len(sentence)-5

sentence = sentence[8:length]

print(sentence)

sentence = regex.sub('', sentence)

for word,pos in nltk.pos\_tag(nltk.word\_tokenize(str(sentence))):

if (pos == 'NN' or pos == 'NNP' or pos == 'NNS' or pos == 'NNPS'):

word = getUpper(word)

if word not in attributes:

attributes.append(word.lower())

text.insert(END,"Extracted Main Attributes From Image: "+str(attributes)+"\n")

def upload(): #function to upload tweeter profile

global filename

text.delete('1.0', END)

filename = filedialog.askopenfilename(initialdir="imgs")

text.delete('1.0', END)

text.insert(END,filename+" loaded\n");

def existing():

global existing\_correct

attributes.clear()

mytags.clear()

G7 = nx.read\_pajek('data/img7.net')

[annotators,tags] = nx.bipartite.sets(G7)

for val in tags:

if len(attributes) < 15:

attributes.append(val)

for val in tags:

mytags.append(val)

text.delete('1.0', END)

text.insert(END,"Tags for image 7\n\n");

text.insert(END,str(list(sorted(tags))))

existing\_correct = 0

G7 = nx.DiGraph(G7)

[h7,a7] = nx.hits(G7)

sorted\_a7 = sorted(a7.items(),key=operator.itemgetter(1), reverse=True)

text.delete('1.0', END)

text.insert(END,filename+" loaded\n");

for i in range(0,8):

data = sorted\_a7[i];

if data[0] in tags:

existing\_correct = existing\_correct + 1

text.insert(END,"Existing Correct Annotation : "+data[0]+"\n")

text.insert(END,"Existing technique Correctly Found Annotation : "+str(existing\_correct)+"\n\n\n")

def extension():

global extension\_correct

text.delete('1.0', END)

RCNN(filename)

temp = []

extension\_correct = 0

for i in range(len(attributes)):

for syn in wordnet.synsets(attributes[i].lower()):

for l in syn.lemmas():

if l.name() in mytags and l.name not in temp:

temp.append(l.name)

extension\_correct = extension\_correct + 1

text.insert(END,"Extension Correct Annotation : "+l.name()+"\n")

if attributes[i] in mytags:

extension\_correct = extension\_correct + 1

text.insert(END,"Extension Correct Annotation : "+attributes[i]+"\n")

text.insert(END,"Extension technique Correctly Found Annotation : "+str(extension\_correct)+"\n\n\n")

def graph():

height = [existing\_correct,extension\_correct]

bars = ('Existing Correct Annotation', 'Extension Correct Annotation')

y\_pos = np.arange(len(bars))

plt.bar(y\_pos, height)

plt.xticks(y\_pos, bars)

plt.show()

font = ('times', 16, 'bold')

title = Label(main, text='Filtering Instagram Hashtags Through Crowdtagging and the HITS Algorithm')

title.config(bg='firebrick4', fg='dodger blue')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=0,y=5)

font1 = ('times', 12, 'bold')

text=Text(main,height=20,width=150)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=50,y=120)

text.config(font=font1)

font1 = ('times', 14, 'bold')

uploadButton = Button(main, text="Upload Image", command=upload, bg='#ffb3fe')

uploadButton.place(x=50,y=550)

uploadButton.config(font=font1)

modelButton = Button(main, text="Run Existing Technique & Get Annotate Rank", command=existing, bg='#ffb3fe')

modelButton.place(x=250,y=550)

modelButton.config(font=font1)

runforest = Button(main, text="Run Extension Technique & Get Automatic Sentence & Annotation", command=extension, bg='#ffb3fe')

runforest.place(x=50,y=600)

runforest.config(font=font1)

rundcnn = Button(main, text="Comparison Graph", command=graph, bg='#ffb3fe')

rundcnn.place(x=650,y=600)

rundcnn.config(font=font1)

main.config(bg='LightSalmon3')

main.mainloop()