Performing Text Analysis on Tweets

Introduction

We want to create a flexible framework for analysing a collection of tweets. For now, we want to do the following:

First, we will make a simple webpage which contains a form asking for a Twitter handle. We will then perform some text analysis of, say, the last 1000 of tweets from this handle. For now, the analysis will consist of the most common hashtags used by the handle, and the most common words used in the text of the tweet. For the latter, we will exclude common English words like articles and also collapse different forms of the same word (like “going” and “gone”).

The rest of the document lays down the steps to achieve our objective.

Basic framework

We use Python for coding; the website is created using the Django framework, and the text analysis is carried out with using the nltk library. We use the Twitter API for accessing the Twitter feed and extract metadata from it, and some nltk modules for tweet analysis.

We deploy the code on a server using Google Cloud, Heroku etc. (TODO).

Using Django

Django is a Python framework for web development. We will cover all the required steps in this document; Django also has a step-by-step [tutorial](https://docs.djangoproject.com/en/3.1/intro/tutorial01/) on its website. We assume that Django is already installed. If not, here are the [directions](https://docs.djangoproject.com/en/3.1/intro/install/) to install it. All the code in this document, including this documentation file, is available on github at [this link](https://github.com/belrogue/tweet_analysis).

Creating a Django project

In the main working directory:

django-admin startproject tweet\_analysis

This step will create a project folder named tweet\_analysis in your working directory.

Creating Django app enter\_handle

Now we create an app. We first show the user a form which takes a Twitter handle as input. We then access the handle’s Twitter feed, extract the last 1000 tweets, and then perform text analysis on the tweets. Since the Twitter API part is not completed yet, for demonstration purposes, we use a corpus of Tweets in the nltk library instead.

To create the app:

python manage.py startapp enter\_handle

This step creates a directory called enter\_handle under the project folder tweet\_analysis. (A project is a collection of apps, and an app can be reused in multiple projects.)

Setting up the URL for the input page

First we edit the project-level file tweet\_analysis/urls.py

from django.contrib import admin

from django.urls import include, path

urlpatterns = [

path('enter\_handle/', include('enter\_handle.urls')),

path('admin/', admin.site.urls),

]

Then, we edit the app file enter\_handle/urls.py

from django.urls import path

from . import views

app\_name = 'enter\_handle'

urlpatterns = [

path('', views.get\_handle, name='get\_handle'),

]

We now need to create a web server to handle requests. Django allows us to create a server on our local machine using the following command in the main project directory:

python manage.py runserver

We can now access the URL: <http://localhost:8000/enter_handle/>

However, accessing the URL will lead to an error message, because we have not yet written any function to handle the request. We next show how to do this step.

Working with Forms and Views

We now show how to write a “form” and a “view” which handles the webpage request. First we define a “form”, by editing enter\_handle/forms.py

from django import forms

class HandleForm(forms.Form):

handle = forms.CharField(label='Enter Twitter Handle', max\_length=100)

We then create a “template” to handle the HTML code for the page. Create a directory templates in the app folder and create a subdirectory enter\_handle within it. Create a file: templates/enter\_handle/handle.html

<form action="/enter\_handle/" method="post">

{% csrf\_token %}

{{ form }}

<input type="submit" value="Submit">

</form>

The above two steps describe a simple form with a single textbox for input and a “Submit” button.

We then write a “view” to handle the form request. Here’s the basic template, with all the details removed, to illustrate the ideas. The comments in the code are self-explanatory.

from django.http import HttpResponseRedirect

from django.shortcuts import render

from .forms import HandleForm

def get\_handle(request):

# if this is a POST request we need to process the form data

if request.method == 'POST':

# create a form instance and populate it with data from the request:

form = HandleForm(request.POST)

# check whether it's valid:

if form.is\_valid():

# process the data in form.cleaned\_data as required

# ...

# redirect to a new URL:

return render(request, 'enter\_handle/analyze\_tweets.html', context)

# if a GET (or any other method) we'll create a blank form

else:

form = HandleForm()

return render(request, 'enter\_handle/handle .html', {'form': form})

Using nltk to analyze the tweets

nltk is a Python library for natural language processing. We also use a library called pandas to handle datasets. The following commented code, mostly self-explanatory, shows how to process the form data.

from django.shortcuts import render

from django.http import HttpResponse

from .forms import HandleForm

import nltk

from nltk.corpus import twitter\_samples

from nltk.twitter.common import json2csv

from nltk.tokenize import TweetTokenizer

import pandas as pd

from string import punctuation

def normalization(word\_list):

lem = nltk.WordNetLemmatizer()

normalized\_word = []

for word in word\_list:

normalized\_text = lem.lemmatize(word,'v')

normalized\_word.append(normalized\_text)

return normalized\_word

def index(request):

return HttpResponse("Hello, world. You're at the enter\_handle index.")

def get\_handle(request):

# if this is a POST request we need to process the form data

if request.method == 'POST':

# create a form instance and populate it with data from the request:

form = HandleForm(request.POST)

# check whether it's valid:

if form.is\_valid():

# process the data in form.cleaned\_data as required

# For now, we use the twitter "userid" instead of

## the twitter handle. User ID to Handle conversions can

## be done at https://tweeterid.com/

handle = int(form.cleaned\_data['handle'])

# Use a sample from the Twitter corpus

input\_file = twitter\_samples.abspath("tweets.20150430-223406.json")

# Convert the relevant fields in the json to a CSV text file

with open(input\_file) as fp:

json2csv(fp, 'tweets\_text.csv',

['created\_at', 'favorite\_count', 'id', 'in\_reply\_to\_status\_id',

'in\_reply\_to\_user\_id', 'retweet\_count', 'retweeted',

'text', 'truncated', 'user.id'])

# Read the CSV into a Pandas dataframe

tweets = pd.read\_csv('tweets\_text.csv', index\_col=2, header=0, encoding="utf8")

# nltk stuff

# Only look at tweets from a specific user

tweets = tweets.loc[tweets['user.id'] == handle]['text']

# Convert the tweets into a text string.

raw\_text1 = ""

for tweet in tweets:

raw\_text1 = raw\_text1 + tweet + "\n"

# Tokenize the tweets into separate words, hashtags etc.

tknzr = TweetTokenizer()

text1 = nltk.Text(tknzr.tokenize(raw\_text1))

# Stopwords are common English words which are to be ignored

stopwords = nltk.corpus.stopwords.words('english')

stopwords.append('â€¦')

stopwords.append('rt')

stopwords.append('')

# Lemmatize the text

nom\_text1 = normalization(text1)

# Strip punctuation and stopwords, and compute a frequency distribution of all the words.

allWordExceptStopDistAndNoms = nltk.FreqDist(

w.lower().rstrip(punctuation)

for w in nom\_text1 if

w[0] != '#' and w.lower() not in stopwords

and len(w) > 1)

# Hashtags are just words which start with "#'

all\_hashtags = nltk.FreqDist(w.lower().rstrip(punctuation) for w in text1 if w.lower() not in stopwords and w[0] == '#')

# back to Django

# bundle all the data into a "context" to send

context = {'handle': str(handle),

'tweet\_list': tweets.tolist(),

'most\_common\_hashtags': all\_hashtags.most\_common(10),

'most\_common\_words': allWordExceptStopDistAndNoms.most\_common(10)}

return render(request, 'enter\_handle/analyze\_tweets.html', context)

# if a GET (or any other method) we'll create a blank form

else:

form = HandleForm()

return render(request, 'enter\_handle/handle.html', {'form': form})

Finally, we create a template for the file analyze\_tweets.html, which displays the results of the analysis above. Create a file templates/enter\_handle/analyse\_tweets.html:

<style>

table, th, td {

border: 1px solid black;

border-collapse: collapse;

}

</style>

<h3>Tweets by {{handle}}</h3>

{% if tweet\_list %}

<table>

<th>Tweet</th>

{% for tweet in tweet\_list %}

<tr> <td>{{ tweet}}</td> </tr>

{% endfor %}

</table>

{% else %}

<p>No tweets are available.</p>

{% endif %}

<h3>Most common hashtags</h3>

<table>

<tr>

<th>Hashtag</th>

<th>Frequency</th>

</tr>

{% for word in most\_common\_hashtags %}

<tr>

<td>{{ word.0}}</td>

<td>{{ word.1}}</td>

</tr>

{% endfor %}

</table>

<h3>Most common words</h3>

<table>

<tr>

<th>Word</th>

<th>Frequency</th>

</tr>

{% for word in most\_common\_words %}

<tr>

<td>{{ word.0}}</td>

<td>{{ word.1}}</td>

</tr>

{% endfor %}

</table>

Note that when we “render” the analyze\_tweets page in views.py we send a Python dictionary named context which bundles all the data we want to render on the page.