Facebook Public Data Analysis and Classification Using CNN

Wenbo Sun,Jianan Wen, Zixuan Xu

sun.wenb@husky.neu.edu

wen.ji@husky.neu.edu

Xu.zix@husky.neu.edu

INFO7390 Spring , Northeastern University

Abstract

Social media has become a very significant open communication medium currently. This has motivated a lot research on social media data analysis from different aspects using machine learning techniques. Measuring social media analytics requires a comprehensive strategy, we need to see how engagement and click-thrus on social media, along with website engagement and sales data, all relate to one another in order to connect the dots between content and conversion. Consider the problem of evaluating an Facebook official account, our group frame the problem as collecting dataset and model training. During the progress we extract more than 30,000 pieces of data from 10 public accounts. On the other hands we propose a model combining CNN (convolutional neutral network), correlation and classification to extract features and patterns related social media posts and use them to derive insights about the sources and official account who generated the post with 89.10% accuracy on test dataset. Realize the program's initial understanding of a public homepage and be able to make further predictions.

Keywords: CNN, social media, public data

Introduction

A vast variety of posts including text, images and links occur around the world every second. The diversity and huge amount of these data cause challenges not only for us to collect, but also to classify.

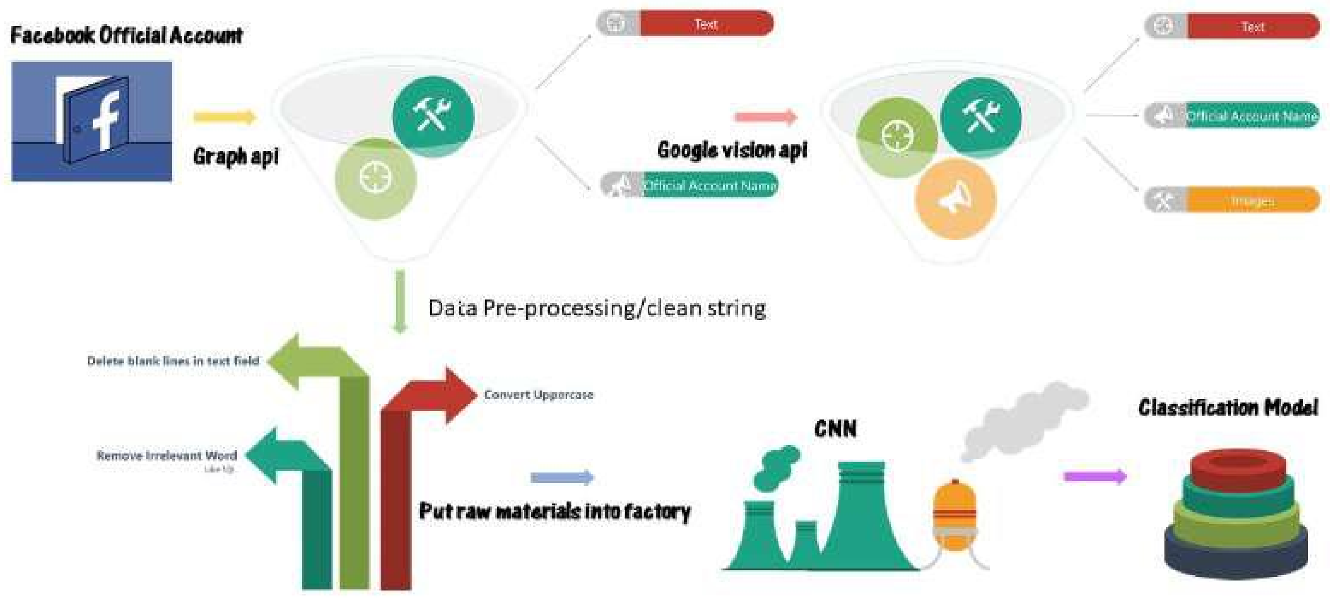
Since Facebook has a very strict rule among user privacy policy, we can only obtain information published in our own account or information released by official account. Therefore we'll only focus on public data.

Utilizing Facebook graph API services to collect data from different official accounts. The raw data we collected consists of two parts: text and account name. In this part we also need access a token to access graph API. Original dataset, as known as target came serialized in JSON which we acquired by a three-step loop and then performed all exploratory data from this section. In fact, there exist a huge amount of posts are merely forwarded and do not have their own textual information. In this case we forced to jump out of loop to maintain data acquisition.

A same method is also used to process pictures. We use google vision API to deal with image content. It provides powerful Image analytics capabilities and enables us to build the next processing. API can see and understand the content within the images and finally return key words which can describe the images forward us.

To make data suitable for further classify and analysis, before convolutional neutral network digesting we completed bellowing pre-processing:1.Converted all characters to lower cases. 2.Remove punctuations and irrelevant symbols. 3.Remove all blanks via text field.

Integrate pre-processed text information into a csv file. This csv file contains label and text and shown as 'industrial raw material' in the picture below. Processing through 'factory-CNN' we will reach classification model.



The overall flow chart

Code and Documentation

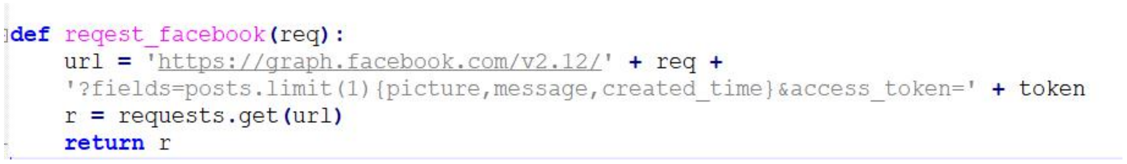
In this paper, we will only talk about the core of code. Please check our portfolio for the details of our whole code.

Part A: Creating data

1.Facebook graph API

There could be thousands posts on Facebook every minute. So, it is really pointless to work on a old dataset. The first step of our project is creating our own dataset.

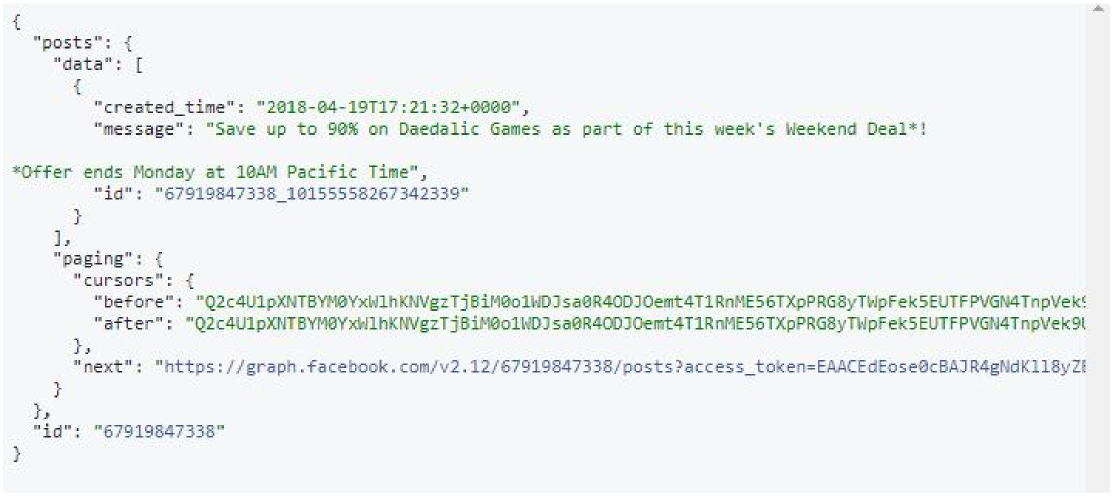
We are using Facebook graph API to get the fresh posts from Facebook(Check this link for how to use Facebook graph API: https://developers.facebook.com/docs/graph-api).



```
def reqest_facebook(req):
    url = 'https://graph.facebook.com/v2.12/' + req +
    '?fields=posts.limit(1){picture,message,created_time}&access_token=' + token
    r = requests.get(url)
    return r
```

This is the code we used to get data from a post. The "req" is the target you want. And you will need a token to access to the API(check this link for how to get a token: https://towardsdatascience.com/how-to-use-facebook-graph-api-and-extract-data-using-python-1839e19d6999)

The we can easily transform the "r" that return by the previous method to json.

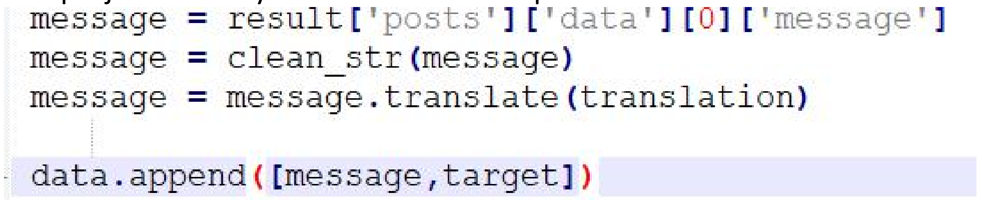


```
{
  "posts": {
    "data": [
      {
        "created_time": "2018-04-19T17:21:32+0000",
        "message": "Save up to 90% on Daedalic Games as part of this week's Weekend Deal*!

*Offer ends Monday at 10AM Pacific Time",
        "id": "67919847338_10155558267342339"
      }
    ],
    "paging": {
      "cursors": {
        "before": "Q2c4U1pXNTBYM0YxWlhKNVgzTjBiM0o1WDJsa0R4ODJOemt4T1RnME56TXpPRG8yTWpFek5EUTFPVGN4TnpVek
        "after": "Q2c4U1pXNTBYM0YxWlhKNVgzTjBiM0o1WDJsa0R4ODJOemt4T1RnME56TXpPRG8yTWpFek5EUTFPVGN4TnpVek9
      },
      "next": "https://graph.facebook.com/v2.12/67919847338/posts?access_token=EAACEdEose0cBAJR4gNdKl18yZ
    }
  },
  "id": "67919847338"
}
```

This is how does the json looks like. We can get the information like, time, text, url to the image, or we can go to check the next post.

Our project will only focus on the text in post.



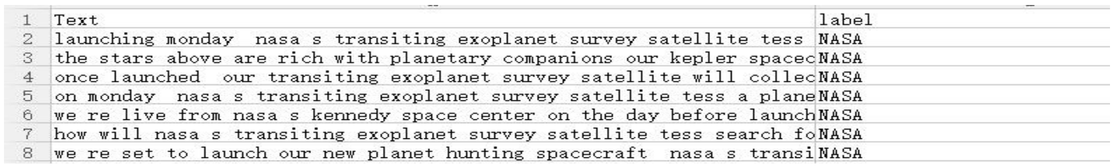
```
message = result['posts']['data'][0]['message']
message = clean_str(message)
message = message.translate(translation)

data.append([message,target])
```

This code will create a list that contain the text and account name.

2.Dataset

This is how our data looks like:



Text	label
launching monday nasa s transiting exoplanet survey satellite tess	NASA
the stars above are rich with planetary companions our kepler spacec	NASA
once launched our transiting exoplanet survey satellite will collec	NASA
on monday nasa s transiting exoplanet survey satellite tess a plane	NASA
we re live from nasa s kennedy space center on the day before launch	NASA
how will nasa s transiting exoplanet survey satellite tess search fo	NASA
we re set to launch our new planet hunting spacecraft nasa s transi	NASA

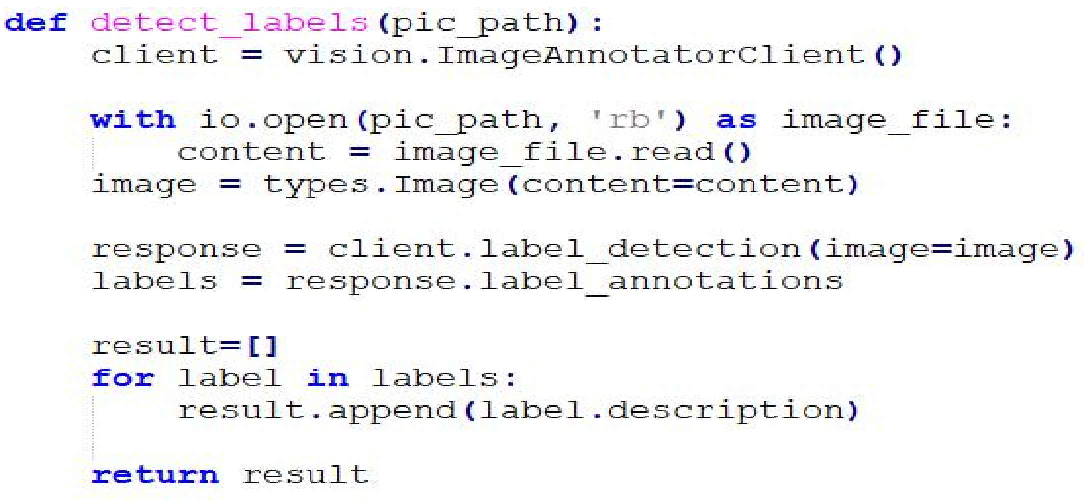
After couples of hours running, we totally have 30 thousands lines of data from more than 10 public accounts.

We are going to use that data to build a machine learning model.

3.Processing pictures

We also have a function to process pictures in a post and transform them to keywords. We used Google vision API to do this. However, process picture take a very long time. We are not going to using this functionality at this time. But maybe we will add more features in the future. So we keep that code in our project.

Check this link for how to use Google vision API: https://cloud.google.com/vision



```
def detect_labels(pic_path):
    client = vision.ImageAnnotatorClient()

    with io.open(pic_path, 'rb') as image_file:
        content = image_file.read()
    image = types.Image(content=content)

    response = client.label_detection(image=image)
    labels = response.label_annotations

    result=[]
    for label in labels:
        result.append(label.description)

    return result
```

This method will take a path of a picture and return a list of keywords.

Google vision API can use a picture url. But the picture url on Facebook are in a special format. We can not use it directly. So we will download the picture to local and process it then delete it.

Part B: Building Model

1.Pre-processing and structure design

Now we have our dataset ready. The next step is to build a machine learning model on that. The text in our dataset is one or many human readable sentences. So we choose to use CNN(convolutional neural networks).

The CNN model we are using is form http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/

This model by DENNY BRITZ is licensed under the Apache License Version 2.0 https://www.apache.org/licenses/LICENSE-2.0

The input layer is depend on the max text length in dataset. We use word2vector to transform a single word to a vector.

We also need to transform the label to a vector as well. This vector will be our output layer. The size of this vector is equal to the number of label. For example, if we have “google”, “steam”, and “att”, the output vector size is 3.

Because building a CNN with a long output size required too much memory and time. We will only use five public accounts data. If it still too slow on you machine, you only load three accounts data.

Use load_data_and_labels5 for load 5 accounts, load_data_and_labels3 for 3 accounts.

After processing data, we are ready to build the CNN model.

We have some configuration parameters:

act_function: 0 for relu, 1 for elu

cost_function: 0 for softmax_cross_entropy, 1 for sigmoid_cross_entropy_with_logits

gradient: 0 for Adam, 1 for Adagrad

epoches: number of epochs

architecture: embedding dimensions for word2vertor and number of filters for CNN

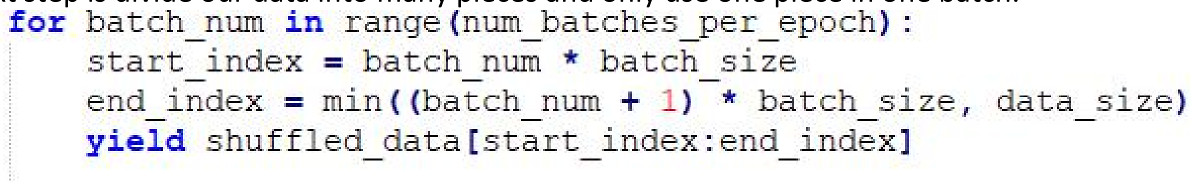
filter_sizes: size for each filters in CNN init: initialization config, True for uniform, False for random

We can try different settings and see which one do the best job.

Then we build vocabulary based on the input data, and use that vocabulary for word2vector embedding.

2.Training

Next step is divide our data into many pieces and only use one piece in one batch.



```
for batch_num in range(num_batches_per_epoch):
    start_index = batch_num * batch_size
    end_index = min((batch_num + 1) * batch_size, data_size)
    yield shuffled_data[start_index:end_index]
```

This code will create a list of data for each batch.

In the middle of training, we will record the loss and accuracy for every batches and save the model every 100 batches.

Then we will feed the data into CNN model and train the model.

2. Evaluate model

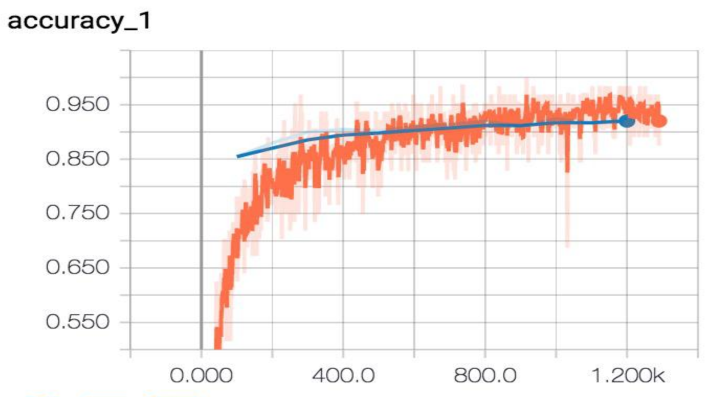
After hours of training, we will have a model and tensorflow will save that model into your project folder. You can evaluate that model and use it.

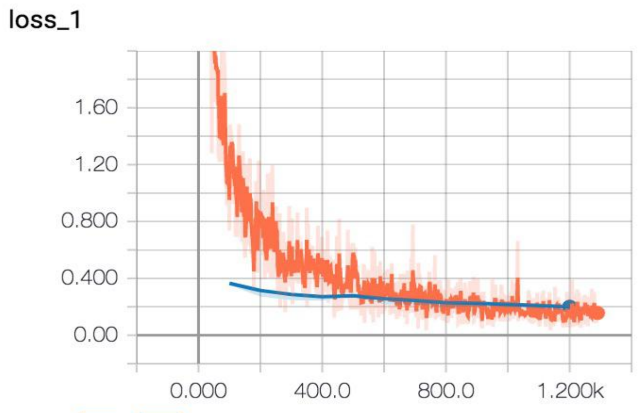
We create a new test dataset to evaluate the model and do prediction on it.

Result:

Finally, we have 5 official account, total posts number is 18293

After 5 epchos, 1290 steps training, the best accuracy in validation set is 89.10%.





We use 1.5 hours for 5 epochs on a 5 classes model. We can easily extend that model by only changing config paramaters.

Conclusion

In this paper we discussed implementing a CNN for text classification in tensorflow. Utilizing Facebook and Google API retrieve raw data, performing variety of methods to pre-processing and classifying. Sorting and batching the data into neural networks.

The model presented in the paper can determine who the publisher is by one former post and achieved good classification performance with 89.10% accuracy across a range of text classification tasks(over 30,000 sentences).

However, the model we build based on CNN mainly focus on sentences which include explicit context and semantic logic rather than a few keywords without obvious relationship. Besides, processing pictures really take a long time. We are sure we can improve our model by spending more time on training.

References：

1. Britz, D. (2015, December 11). Implementing a CNN for Text Classification in TensorFlow. Retrieved from http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/
2. Britz, D. (2015, November 7). Understanding Convolutional Neural Networks for NLP. Retrieved from http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/
3. Kim, J. (2017, December 2). Understanding how Convolutional Neural Network (CNN) perform text classification with word embeddings. Retrieved from https://towardsdatascience.com/understanding-how-convolutional-neural-network-cnn-perform-text-classification-with-word-d2ee64b9dd0b
4. A Beginner's Guide to Deep Convolutional Neural Networks (CNNs). (n.d.). Retrieved from https://deeplearning4j.org/convolutionalnetwork.html
5. Word2Vec word embedding tutorial in Python and TensorFlow. (2017, July 21). Retrieved from http://adventuresinmachinelearning.com/word2vec-tutorial-tensorflow/
6. Python TensorFlow Tutorial – Build a Neural Network. (2017, April 8). Retrieved from http://adventuresinmachinelearning.com/python-tensorflow-tutorial/
7. Gaussic. (2018, February 19). CNN-RNN 中文文本分类，基于 tensorflow. Retrieved from https://github.com/gaussic/text-classification-cnn-rnn