Author: Toni Narciss

Date: August 20, 2019

Documentation Purpose: This file details the logical choices and coding approach used in developing indices 1,2 and 3, it also states the assumptions made. The file manages to cover all the steps needed however it does not go over the code line by line.

Index 1 and 2: Detecting the sentiment variations and hot locations as a function of time using the twitter data

Overview: this graph looks at 2 data frames one relating AI tweets and another relating to DL tweets. Each DF contains data from 2013 till the present. The graph covers the sentiment of the tweets on a weekly basis and shows the locations that are increasingly being mentioned in tweets on a monthly basis.

Note: A lot of the processes listed below(e.g. NER, SA, …) include saving the work to the disk with every loop. This is necessary since these processes are often time consuming and are prone to interruptions due to connection to instance problems, memory limitations, syntax errors and more.

1. Sentiment Analysis

Importing the data

- the data being handled cannot be processed all at once due to memory limitations, so it was chunked by search phrase(e.g. #AI which belongs to the AI DF).
- The glob package was used to loop over the DFs. At this stage the data was imported as a pandas DF while using "usecols=['tweet', 'date/time']" to import only the required columns and save memory.

Filtering Data by Language

- using the cld2 package detect English tweets to be used for sentiment analysis. This is done to remove any bias non-English languages might introduce to the results such as an increase in neutrality.
- The custom function "detm(x)" is used. This function is created to avoid interruptions caused by encoding errors that arise typically when the cld2 package tries to detect certain characters from mandarin or emojis.
- if the detm() function returns English('en') then use the tweet, if it returns any other language or "error" then skip.

Clean Data

- using the **"**clean_tweet(tweet)" function clean each tweet. This is done to ensure that the sentiment analysis is not biased by links and special characters. This function is included within the analyze_sentiment(tweet) function.

Analyze Sentiment:

- using the "analize_sentiment(tweet)" function identify the sentiment of the cleaned English tweet. The function uses the sentiment.polarity function in the Textblob package to return a holistic score of the tweet's sentiment on a continuous scale from -1 to 1 where -1 is very negative, 0 is neutral and 1 is very positive. The function chunks the results into 3 groups all negative is classified as -1, 0 is kept as 0 representing neutral and all positive scores are classified as 1.

- Using the analyze_sentiment function in a list comprehension loop over each tweet and add the results as a new column labeled “SA” to the DF so that a link between the tweet date and sentiment is established. This is later used to detect how much of each week is positive, neutral or negative.
- Save the data produced as a csv with three columns: tweet, date/time, and SA.

Produce Weekly Sentiment

- Use a dask dataframe that combines all the above produced dataframes. This is needed because the desired dataframe must include all the sub-dataframes(#AI, #NLP, …). Dask is essential since it allows for parallel computing and saves a considerable amount of memory. Using a pandas DF is not an option due to memory limitations. Parallel computing is set using a command at the beginning of the script(“dask.config.set(scheduler='multiprocessing')”)
- To produce weekly sentiment, choose a start date. In this case “2012-12-30” is chosen since it is the Monday of the first week of 2013. Use a while loop that keeps going till the most recent date in the dataframe is reached.
- In each loop take the next 1 week slice of the dask DF, where “since” is the beginning of 1 week and “until” is the beginning of the next week.
  one_weeks_data = data.loc[(data['date/time']>=since) & (data['date/time']<until),:]
- Use the datetime package to easily manage dates. A datetime timestamp is necessary to be able to add 1 week to the since and until variables with each loop. Also, It is used to switch a datetime timestamp from the starting string. In addition, It is used to produce a string from the datetime timestamp. Comparing dates, although inelegant, is done while the dates are in string format for ease of use and transferability via file types.
- Use the *one_weeks_data* slice and measure the length of the 3 slices of the SA column(-1, 0,and 1). Then produce the ratio of each slice’s length to the total length of the week’s data. And save the results in a dataframe. Once the loop is over save the dataframe as a csv. Multiply the resulting ratios by 100 to turn them into percentiles and then round them to 1 decimal place since more decimal points are not significant in sentiment analysis.
- The resulting df is called weekly_df and contains 4 columns, the first called “wc” (week commencing) and the rest “SA_pos”, “SA_neu”, and “SA_neg” respectively.

2. Location Identification

The While Loop

- similar to the while loop above in part 1, except in this part slice that data into 1-month slices. Use a 1-month slice here to get a large enough dataset. the logic is similar to the above part except here use a different package for adding the timedelta each month called dateutil since datetime's timedetla does not contain "month" as a time unit.

Collapse the Dask DF slice

- use the Dask DF to create a temporary DF (one_months_data)which contains data of the month of interest. Change the one_months_data from a dask DF to a pandas DF using the .compute() method. This is required in order to be able to pass certain functions onto the tweets in each instance such as tweet cleaning and named entity recognition(NER).

Clean data

- using a list comprehension collapse the Dask DF and loop over the tweets to clean them and create a list of clean tweets to be used for NER. The same clean_tweet(tweet) function used in part 1 before SA is used here as well.

Location Identification

- use Spacy's Named Entity Recognition function to identify locations within tweets.
- Start by importing Spacy's largest English language model('en_core_web_lg') to ensure the best accuracy possible. While importing the language model, using Spacy's load function, turn off unnecessary functionalities in it to significantly boost NER speed as follows: disable=['parser', 'tagger', 'textcat'].
- While cleaning the data each word in each tweet is capitalized in preparation for NER. This is necessary since spacy fails to identify a lot of entities when they do not have a capital first letter.
- Use a list comprehension to create a list of locations found in each tweet. One list for each month, then pickle and save the list to the disk with each loop.

Spike Identification

- create a counter for 2 consecutive months(x1 and x0), the month being studied and the previous month(hence first month being studied is 2013-02). The counter counts the frequency of occurrence of each city/location within each month.
- If the city available in x1 is also available in x0 then it is considered. This is done to avoid using wrongly tagged cities that are available in abundance within 1 month.
- If the ratio of x1/x0 > 12 then this city has increased in occurrence by 1200% then use the location. This threshold provides results of 3 to 20 cities/locations per month. These values however need manual cleaning, typically 50%-90% of the phrases tagged are correct.
- Compile the resulting city alongside the corresponding percentile increase('London: 2400%') to the list of spiking locations in that month.
- Create a DF (city_df) for the locations, the DF has 2 columns: the months which is set as an index, and the spiking locations.

3. SA and Location Visualization

Data Preparation

The bokeh line charts have a datetime x-axis. The SA data has weekly intervals whereas the location data has monthly intervals, therefor some preprocessing is needed.

- Import the weekly_df which contains the weekly sentiments.
- Then add a new column(‘cities’) that adds the monthly locations to the weekly sentiment DF. This is done by duplicating the results of the monthly locations for every week covered in the weekly sentiment DF(e.g. week 2013-03-05 and 2013-03-12 would both get the results of 2013-03 from the monthly locations df: city_df)

Plotting

Now that a DF is ready with both sentiment and locations sharing a common time index plotting can be done using bokeh.

- The Python package Bokeh is chosen because it can leverage hover tools and a dynamic use of the datetime data in the x-axis. This allows for a more data to be presented with little ink being added to the graph. The downside that should be mentioned is that a dynamic chart might increase in the complexity of the graph.
- The bokeh figure should be initialized by specifying the y-axis which represents the percentiles as linear and the x-axis representing the datetime as datetime. As mentioned above this is necessary to ensure that bokeh understands the type of data is being fed and allow the user to use the functionalities that accompany that graphing choice.
- The data is plotted in two chunks the first is the total line which shows information relating to location spikes as a function of time while the second chunk are the 3 sentiment lines which show the positive, neutral and negative sentiment respectively as a function of time.
- Chunking is needed in order to ensure distinct features within each hover box that appear over each line.
- This can be achieved via an inelegant solution which plots 4 different layers of hover tools(this is a temporary workaround solution for this documented problem in bokeh’s GitHub page). As a result each layer will have its own on/off hover tool switch on the side of the graph enabling the users to toggle this feature as they please.

Index 3: Detecting Top 100 Companies per month in twitter data

Process overview: detect companies mentioned in tweets using Spacy's named entity recognition function, create and use a logistic regression classifier based on the results of google searches to filter out the entities that are truly companies and then display the results of the top 100 companies per month in a table format, with a table for each month.

1. Company Detection using NER

Define Functions

- **clean_tweet(tweet):** this function takes one string/tweet at a time and cleans it. It removes most links, all digits and special characters.
- **save_pickle(path, list_):** this function takes only a path and a list. It is a simple function meant to save lists as pickled files. It is used in the file below to export monthly companies detected using spacy's NER to the disk.
- **load_pickle(path):** similarly, this function takes the path, and it is meant to load saved pickled lists.
- **list_length(path):** this function takes the path of the list desired. It checks if the list exists, returning 0 if it doesn't and the list's length if it does. This is needed since the NER process over this amount of data is lengthy and is prone to interruptions; this function is used to determine the latest checkpoint in the pickled files.

Import Data

- Create a dask dataframe that will be used to create a list of tweets for each month to be used for NER. Import the data in as 2 DFs: AI and DL. These DFs are a combination of multiple sub-DFs (e.g. AI contains #AI, #NLP,…) and the AI DF is too large to be loaded on the instance's memory. Thus, Dask is needed to slice it into monthly data before processing.
- While importing the data make sure to import only the 2 columns needed: "tweet" and "date/time"

NER

- Set start and end date varibales for the slicing process. These variables are to be used for slicing the DF into monthly chunks. They are updated each loop using the for loop conditions. They are strings and are kepts that way when being used as conditions for slicing for efficiency and simplicity purposes; changing them to datetime variables would add complexity especially when dealing with dask DFs.
- Slice the dask DF into monthly chunks and collapse each chunk temporarily in the loop using the .compute() method of the dask DF. Then keep only the tweets of that month as a python list to minimize memory usage.

- Using a list comprehension and the clean_tweet function clean the tweets then pass the .title() method onto each cleaned tweet to ensure the results have the first letter of each word capitalized. This is necessary to ensure the best results from spacy's NER.
- Use spacy to conduct NER. Import Spacy's largest English language model for best results("en_core_web_lg"). in addition, disable the unnecessary features in the language model to significantly speed up the NER process(disable=['parser', 'tagger', 'textcat']).
- Drop companies that appear only once per month. This is done to remove insignificant companies and also this assumes that the vast majority of entities recognized with frequency=1 are incorrectly tagged by spacy. This assumption is based on several trials . furthermore, since the results of this index are focused on the top 100 companies, entities with frequency=1 are irrelevant.
- Save data as pickled lists with every loop to the disk to avoid loss of results due to interruptions to the lengthy loop.

2. Filtering NER Results using Logistic Regression

Define Functions

- **clean_text(list_of_strings):** this functins takes a list of strings and outputs the same format. using the preprocessing package's clean function, each tweet is cleaned for links, special characters and numbers. Then all data is transformed to lowercase to ensure homogeneity during classification.
- **features_frequency_list(list_of_features, list_of_descriptions):** this function takes a list of features as strings and a list of descriptions from each google result also as strings. Before searching in the list of descriptions provided the clean_text function is passed. It returns how often each feature occurred in each google search. It returns a list integer representing the frequency of each feature in order.

Train a logistic model

- get classified data split 50/50 where half are companies downloaded from the UK government's recently opened companies and the other half are entities wrongly classified by spacy as companies. For the purpose of this index the data contained 3000 instances.
- Use this data to create a simple logistic regression model.
- The features of the model are : 'company', 'startup', 'incubator', 'corporation', 'multinational', 'accelerator', 'business', 'linkedin', 'funding', 'indeed', 'venture', and a final created one 'total'.
- Each feature is the amount of time the word is mentioned in the descriptions of all the results appearing on the first page of the google.co.uk search page. the total feature is the sum of all the features. The total feature was created as an intuitive holistic feature which improves the accuracy of the model. This is done by using features_frequency_list function.
- The model is kept at its default parameters without any fine tuning because the results reach 90% accuracy and weighted F1 score. However, a a gridsearch can be conducted to improve the

accuracy score. (Note: since the final results must be 100% accurate some manual classification/cleaning will always be needed.)

- Save the resulting logistic model.

3. Scraping Google

Scraping Google.co.uk Search Results

As a starting note it is worth mentioning that scraping google is tricky and has time limitations. As a result, scrapes can be done consistently in batches of 300 searches at a time. For each batch a new IP is required, and the IPs being used should be clean(not blocked previously by google). Therefor, scraping should be kept at a minimum.

- From the pickled lists of detected companies select the top 1500 per month. From these 1500 typically the top 900 companies are scraped. This typically results in 150-200 companies that are get classified as companies.
- Each company gets a google link search in preparation for the search. Then the links are passed to a scrapy spider in batches of 300.
- The results of the scrapes are saved as csv files.

Classifying the Google Search Results

- The resulting csv files are used to classify the companies using the logistic model.
- The entities classified as companies are kept.

4. The Visualization

- Create a DF that contains those companies, the corresponding month, and the frequency count.
- Sort the results by frequency count in descending order
- Create a column that shows a change in rank between the current month and the previous month. If a company is new display “NEW”.