ODU Big Data, Data Wrangling Boot Camp
Software Overview and Design

Chuck Cartledge

January 17, 2017

Contents

List of Tables

List of Figures

List of Algorithms

1 Introduction

The tweet sentiment analysis software used as part of the Old Dominion University College of Continuing Education and Professional Development Big Data: Data Wrangling boot camp[1] will be used to provide boot-camp attendees with hands-on experience doing data-wrangling of textual data.

> *"We define such data wrangling as a process of iterative data exploration and transformation that enables analysis. . . . In other words, data wrangling is the process of making data useful."*
>
> Kandel et al. [1]

In the boot-camp, we will be looking at tweets to conduct sentiment analysis relative to arbitrary hashtags. We will be focusing on data wrangling (see Figure 1) using both Python and R programming languages.

Each boot-camp workstation will have the same software load (see Section C), and almost fully functional software in Python and R. The software will be complete in that it will:

- Retrieve tweets from Twitter,
- Place tweets in a PostGres database,
- Retrieve tweets from the database,
- Tokenize the tweets,
- Qualify the tweets as positive, negative, or neutral, and
- Plot the results in different ways.

Data wrangling will focus on:

1. Identifying problems with the tweet tokens,
2. Developing solutions to those problems, and
3. Reducing the number of problematic tokens.

The remaining sections layout in detail the overall system design, details of the major algorithms, database tables, and the configuration file used to control the system.

[1] `https://www.odu.edu/cepd/bootcamps/data-wrangling`

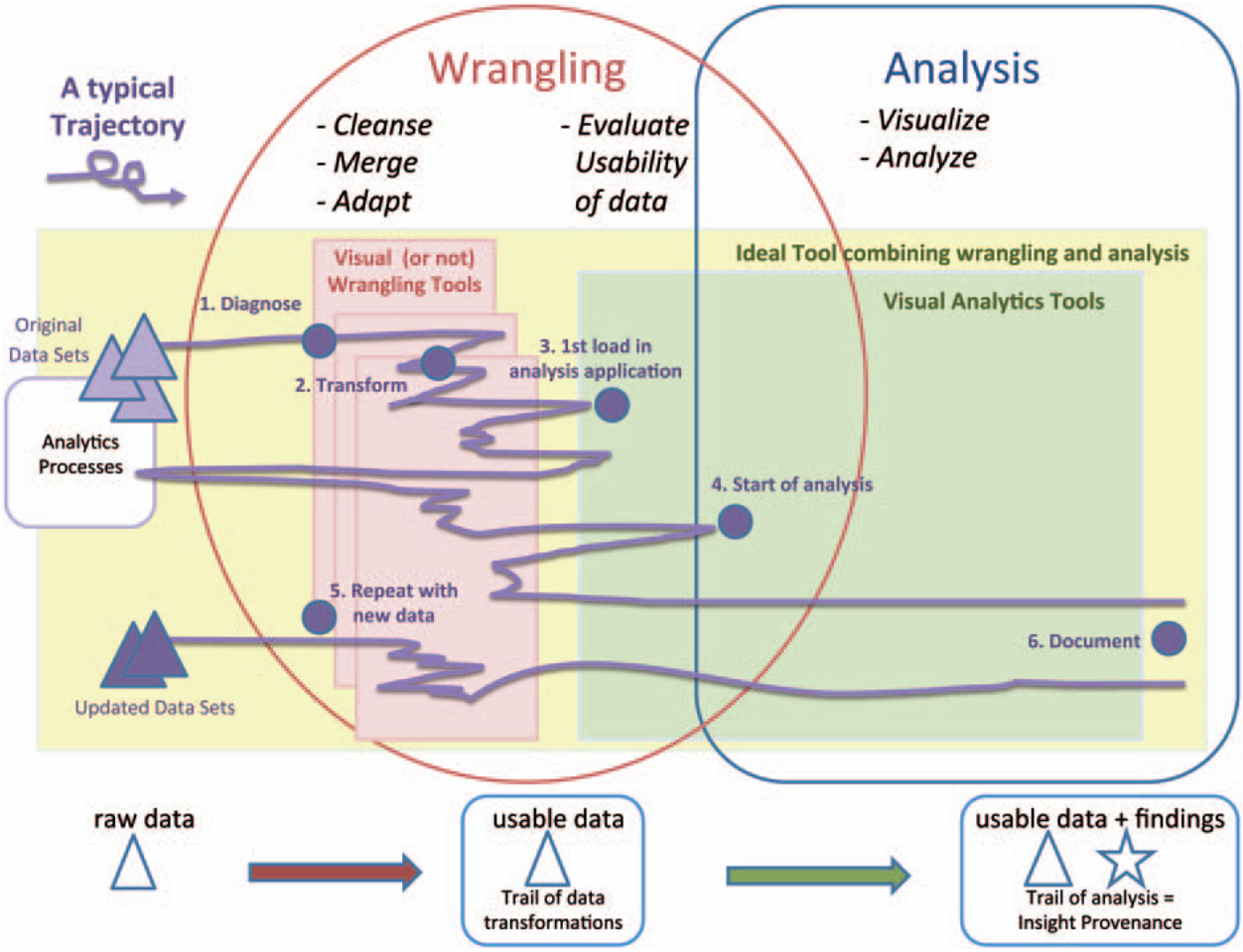

Figure 1: Notional data science data flow. Data wrangling requires domain specific knowledge to cleanse, merge, adapt, and evaluate raw data. Image from [1].

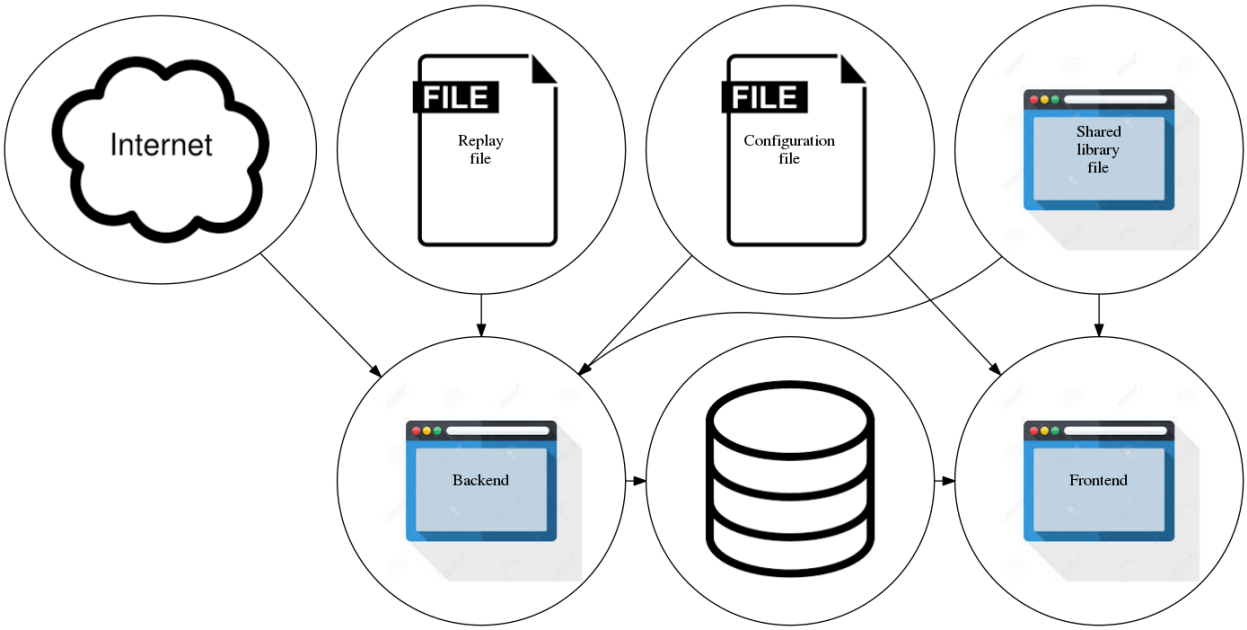

Figure 2: System design. Both front and back ends read data from a common configuration file, and use a shared library file of common functions.. The back end will receive data from the internet or from a replay file, based on directives in the configuration file and update the database with new data. The front end will connect to the database and retrieve data based on directives from the configuration file.

2 Software system design

The sytem is logically divided into three parts (see Figure 2):

1. A "backend" that gets tweets from Twitter or a data file.
2. A database to hold tweets from the backend.
3. A "frontend" that retrieves data from the data base for analysis and display.
4. The frontend and backend processes are controlled by the contents of a configuration file (see Section 2.2).

The backend receives data from Twitter or a data file, puts the data into the database, and the frontend retrieves the data for processing.

2.1 Algorithms used by the software front and back ends

Details of the various algorithms used by the backend and frontend processes are outlined in this section.

Table 1: Frontend and backend algorithm cross matrix. Alogrithms that are used by both the front and back ends are recommended for a "utillity" file or library that can be accessed by both ends.

Num.	Name	Back end	Front end
1	Process configuration file.	✓	✓
2	Update database with new data	✓	
3	Normalize text		✓
4	Evaluate text		✓
5	Update display with new data		✓
6	Plotting hash tag based Tweet sentiment		✓

Input: Location of configuration file
Assign default values;
while *not at the end of the file* **do**
    read line;
    if *not a comment line* **then**
        get token;
        get value;
        if *is a Hashtag* **then**
            add value to list of hashtags;
        else
            structure token value = value;
        end
    end
end
Result: A language specific data structure, values from file override defaults.

Algorithm 1: Process configuration file.

```
Input: Language specific configuration structure
start = first time in data file;
if Offset = TRUE then
    diff = now() - start;
else
    diff = 0
end
time_end = start + SleepyTime + diff;
for Polls remaining do
    if Live then
        submit query to Twitter;
        request data from Twitter;
        for lines from Twitter do
            extract time from JSON;
            data = base64 encoding of entire JSON;
            insert time and data into database;
            if CollectionFile is not NULL then
                append time and data to CollectionFile;
            end
        end
        sleep SleepyTime;
    else
        read line from file;
        parse line into time and data;
        while time > time_end do
            time_end = time_end + SleepyTime;
            sleep SleepyTime;
        end
        insert time and data into database;
    end
end
Result: An updated database.
```

Algorithm 2: Update database with new data.

Input: Text to be “normalized”, “stop word list”
cleansed = Null;
for *Text* **do**
    lower case Text;
    remove non-ASCII;
    stemming;
    if *Text not in “stop word list”* **then**
        append Text to cleansed
    end
end
return *cleansed*;
Result: Normalized text

Algorithm 3: Normalize text.

Input: sourceText, baseLineText
numberOfSourceWords = number of words in baseLineTex;
percentage = numberOfSourceWords / numberOfWordsInSourceText;
return *percentage*;
Result: Percentage of source text in baseLineText

Algorithm 4: Evaluate text.

Input: Language specific configuration structure
cleansedPositive = normalize positive words;
cleansedNegative = normalize negative words;
cleansedStopWords = normalize Stop words;
timeStart = minimum time from database ;
for *Polls remaining* **do**
    timeEnd = timeStart + SleepyTime;
    hash tag new data = NULL;
    lines = query database from timeStart to timeEnd;
    for *lines* **do**
        tweet = base64 decode of data;
        if *parse Tweet is GOOD* **then**
            extract text;
            extract hash tag from Tweet text;
            cleansedText = normalized text less cleansedStopWords;
            positive percentage = evaluate cleansedText vs. cleansedPositive;
            negative percentage = evaluate cleansedText vs. cleansedNegative;
            neutral percentage = 100 - positive percentage - negative percentage;
            update plotting information (hash tag, source, location);
        end
    end
    plot hash tag results;
    timeStart = timeEnd;
    sleep SleepyTime;
end
plot source information;
plot location information;
Result: An updated display.

Algorithm 5: Update display with new data.

Input: The previous/current plotting data, and new data
for *Each Tweet type* **do**
  set the lower left polygon point as the previous poll and the last previous type count;
  set the upper left polygon point as the previous poll and the last previous type count + next previous type count;
  set the lower right polygon point as the current poll and the current type type;
  set the upper right polygon point as the current poll and the current type count + next current type count;
  plot the polygon, filling it with then Tweet type color
end
for *Each Tweet type* **do**
  set previous Tweet count value to current Tweet count value;
end
Result: An updated display data structure, and display.

Algorithm 6: Plotting hash tag based Tweet sentiment. From the user's perspective, a stacked histogram is plotted. From a programatic perspective, each three filled polygons are plotted where the left and right edges are the poll number, and the vertical component is the number of Tweets per type (positive, neutral, and negative). The display will show the absolute number of Tweets, and the color bands will show the proportions of each type.

2.2 Configuration file

The software frontend and backend processes are coordinated by control values in a shared configuration file.

1. A common configuration file to be used by both the data capture and the data presentation programs.
2. The file will default to a "well known" name in a "well known" location.
3. An alternative file can be passed in as a command line argument.
4. Any line in the file starting with a hashtag (#) will be treated as a comment and not processed.
5. File entries are case sensitive.
6. All entries are optional. Some are required for live operation capture.
7. If the same option appears more than once, the last option will be honored, **except for hashtags.** Hashtags will be treated as a collective.
8. "White space" separates each token from its value.

Table 2: Configuration file entries. The default stop word file will be provided (source: `http://xpo6.com/list-of-english-stop-words/`). It can be modified or replaced as needed.

Token	**Meaning**	**Default**
APIPrivateKey	Twitter private API key. Must be supplied for live operation.	(None)
APIPublicKey	Twitter public API key. Must be supplied for live operations.	(None)
CollectionFile	A file to collect raw Tweets during live operations.	(None)
ColorNegative	The color used to indicate negative tweets.	BLACK
ColorNeutral	The color used to indicate neutral tweets.	WHITE
ColorPositive	The color used to indicate positive tweets.	GREEN
Hashtag	This is the hashtag used to search Twitter **without the leading hashtag (#).**	(None)

(Continued on the next page.)

Table 2. (Continued from the previous page.)

Token	Meaning	Default
LexiconFile	A text file containing positive and negative words.	lexicon.csv
Offset	Should the replay data be brought forward to current time? Accepted values are TRUE or FALSE.	FALSE
Poll	How many times to add new data to the database. If data is being replayed, the maximum number of database updates will be this value, or the end of data from the file. If live operations, then this is how many times Twitter will be polled for new data.	10
PostgresTable	The Postgres table containing the tweets.	tweeets
PostgresUser	The Postgres user name used to access the database.	openpg
PostgresPassword	The Postgres password associated with the Postgres user.	new_user_password
ResetDatabase	Should the database be reset, and all previous data lost when the program starts. Accepted values are TRUE or FALSE.	FALSE
SleepyTime	How many seconds between updates to the database. It is possible that no data will be added to the database if there isn't any Twitter activity for a hashtag.	5
SourceFile	The file containing the data to be replayed. If this option is not set, then the operation is assumed to be "live."	(None)
StopwordsFile	The file containing "stop words" that will not be considered in determining positive or negative sentiment.	stopword.txt
ThresholdNegative	The percentage of words in a tweet considered negative for the tweet to be labeled negative.	0.33
ThresholdPositive	The percentage of words in a tweet considered positive for the tweet to be labeled positive.	0.33

(Last page.)

2.3 Design limitations

The current design polls Twitter for new tweets on a periodic basis. The entire list of search hash tags are polled, any returned tweets are stored in the database, and the system "sleeps" for a number of seconds (as per the configuration file). There are a number of factors that affect this processing cycle, including:

1. The number of hash tags being queried. Each poll takes a finite amount of time, even if no tweets are returned, so the more hash tags being queried, the longer it will take to service the complete list of tags.

2. Each tweet becomes a single row in the database. The more tweets that are returned from the query, the longer it takes to update the database with all the tweets.

3. Each tweet has a unique serial number. Each query includes the serial number of the earliest (the one furthest in the past) one of interest in order to get a complete and continuous tweet stream for the hash tag. The earliest acceptable tweet is updated after a successful query.

4. The no-cost query capability is limited to 100 tweets per query. If more than 100 tweets are created between queries, then the polling process will continue to fall further and further behind.

Because of these design and implementation limitations, if tweets are being created faster than the polling process can collect them, then the system will fall further and further behind.

The limitations imposed by a polling interface can be overcome by using a streaming interface[2]. A polling interface was used because it is simple to design, simple to implement, and simple to test. The back-end process could be replaced by a streaming interface without affecting the front-end process.

3 References

[1] Sean Kandel, Jeffrey Heer, Catherine Plaisant, Jessie Kennedy, Frank van Ham, Nathalie Henry Riche, Chris Weaver, Bongshin Lee, Dominique Brodbeck, and Paolo Buono, *Research directions in data wrangling: Visualizations and transformations for usable and credible data*, Information Visualization **10** (2011), no. 4, 271–288.

[2]Python example:`http://bogdanrau.com/blog/collecting-tweets-using-r-and-the-twitter-streaming-api/`, R example:`http://bogdanrau.com/blog/collecting-tweets-using-r-and-the-twitter-streaming-api/`

A Database tables

These are the database tables/data structures to support Twitter analysis:

Table 3: Tables to support Twitter analysis.

Column	Meaning
time	Unix seconds as extracted from the Tweet.
data	Base 64 encoding of the entire JSON Tweet.

B Notational data structures

These are the notational data structures used by the various processes.

Table 4: Notional plotting data structure. A multidimensional structure indexed by hashtag.

Name	Purpose
PositiveTweetSource	A dictionary/hash table to keep track of the number of positive Tweets by software source. This is for the entire polling period.
NegativeTweetSource	A dictionary/hash table to keep track of the number of negative Tweets by software source. This is for the entire polling period.
PositiveTweetLocation	A dictionary/hash table to keep track of the geographic location of a positive Tweet.
NegativeTweetLocation	A dictionary/hash table to keep track of the geographic location of a negative Tweet.

Table 5: Notional plotting data structure. This structure is indexed by hashtag.

Cell	Use
0	Number of positive Tweets.
1	Number of neutral Tweets.
2	Number of negative Tweets.

C Software on each workstation

This section contains the assumptions about the operating system environment, and software load out for each work station.

1. Operating system: Windows 7
2. Database
   (a) Name: PostgresSQL
   (b) Version: 9.5.3
   (c) Source: `http://www.postgresql.org/download/windows/` and `http://www.enterprisedb.com/products-services-training/pgdownload#windows`
   (d) Superuser password: ODUBootcamp
   (e) Misc: It may be necessary to manually start the Postgres server using these commands in a terminal window:

```
cd "\Program Files\PostgreSQL\9.5\bin"

.\pg_ctl -D "c:\Program Files\PostgreSQL\9.5\data" start
```

3. Software
   (a) Python
      i. Interpreter version: 2.7.12
      ii. Available from: `https://www.python.org/downloads/release/python-2712/`
      iii. Modules:
         - base64
         - csv
         - descartes
         - geos
         - json
         - lxml==3.6.0
         - math
         - matplotlib
         - nltk
         - numpy
         - os
         - pickle

- psycopg2
- shapely[3]
- sys
- time
- traceback
- tweepy
- urllib

(b) Java

- Version: Java SE Development Kit 7u79 (assuming Windows 64 bit OS)
- Available from: `http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html`

(c) pgAdmin

- Version: 1.22.1
- Available from: `https://www.pgadmin.org/download/`

(d) R

- Version: 3.3.1
- Available from: `https://cran.r-project.org/bin/windows/base/`
- Packages:
  i. bitops
  ii. DBI
  iii. ggmap
  iv. ggplot2
  v. httr
  vi. jsonlite
  vii. mapdata
  viii. mapplots
  ix. mapproj
  x. maps
  xi. methods
  xii. NLP

[3]The shapely module depends on a number of other modules and can be difficult to install. The best way that I've found to install the module and all its dependencies is to use the Python and hardware architecture specific whl file from `http://www.lfd.uci.edu/~gohlke/pythonlibs/#shapely` After downloading the whl file, it can be installed with using pip, for example:
`pip install Shapely-1.5.17-cp27-cp27m-win_amd64.whl`

xiii. openssl
xiv. RCurl
xv. rjson
xvi. ROAuth
xvii. RPostgreSQL
xviii. SnowballC
xix. streamR
xx. tm

(e) R-Studio

- Vesion: 0.99.903
- Available from: `https://www.rstudio.com/products/rstudio/download/`

(f) Sublime

- Version: build 3114
- Available from: `https://www.sublimetext.com/3`

Requires:

- Anaconda plugin
  - Version: v1.4.4
  - Available from: `http://damnwidget.github.io/anaconda/`

  Anaconda can also be installed using Package Control for Sublime Text 3.
  i. Install Package Control using the Python script found at:
  `https://packagecontrol.io/installation`
  in the Sublime Console window.
  ii. Using the Package Control capability just added to Sublime, install Anaconda using the directions found at:
  `http://damnwidget.github.io/anaconda/`
  basically:
  `Tools` → `Command Palette` → `Install Package` → `anaconda`

The PATH environment variable should be updated to include the location of the R and Python interpreters.