W205 Exercise2 Architecture

This document describes the architecture, dependencies, file structure, and some implementation details of the Twitter streaming application.

Prerequisite
The application can be run in ucbw205_complete_plus_postgres_PY2.7, assuming starting from an instance afresh. All required files and operations can be accessed by checking out https://github.com/maxyan/UCB_MIDS_W205.git, and going to **exercise_2** directory.

Step-by-Step Installation Guide
A step-by-step installation guide of the application can be found in UCB_MIDS_W205/exercise_2/README.txt, after checking out the git repo.

Application Idea
The idea behind the application is simple: to create a streaming application that reads Twitter feeds, parses them and store the words into a Postgres SQL database, to help determine trending subjects.

The application consists of the following parts:

1. A Tweet spout that uses Twitter API and tweepy to stream the feed
2. A Tweet parse bolt that takes a tweet feed and splits into individual words
3. A word count bolt that takes words and updates both the word count internally and to the PostgresSQL db

For completeness and simple analysis, there are also post-process scripts to plot or show the results.

File Structure
Table below describes the file structure.

File	Directory	Description
Tweetwordcount.clj	/exercise_2/EX2Tweetwordcount/topologies	Storm topology of the application
Tweets.py	/exercise_2/EX2Tweetwordcount/src/spouts	Tweet spout for streaming for step 1 above
Parse.py	/exercise_2/EX2Tweetwordcount/src/bolts	Tweet parse bolt for step 2 above
Wordcount.py	/exercise_2/EX2Tweetwordcount/src/bolts	Tweet word count bolt for step 3
Finalresults.py	/exercise_2	Script for showing wordcount for a word, or all words
Histogram.py	/exercise_2	Script for showing all words between two counts
install_dependencies.sh	/exercise_2	Bash script to install all dependencies
install_python_27.sh	/exercise_2	Bash script to install Python 2.7
configure_postgres.sh	/exercise_2	Bash script to configure Postgres properly for the

		application
pg_hba.conf	/exercise_2	Used by configure_postgres.sh to configure Postgres
set_up_db.py	/exercise_2	Python script to set up the table Tweetcount
README.txt	/exercise_2	Step-by-Step guide for installation of the app

There is also a directory /exercise_2/screenshots, which contains some example results of running the application, which will be discussed later.

Major Dependencies
The table below describes all the known dependencies of the application and how they are satisfied.

Dependencies	**Satisfied by**
Java 7	ucbw205_complete_plus_postgres_PY2.7
Postgres SQL v8.4 or higher	ucbw205_complete_plus_postgres_PY2.7
Apache Storm	ucbw205_complete_plus_postgres_PY2.7
Python 2.7	Installed by install_python_27.sh
Virtualenv	Installed by install_python_27.sh
Lein	Installed manually in README.txt
Streamparse	Installed by install_dependencies.sh
Psycopg2	Installed by install_dependencies.sh
Tweepy	Installed by install_dependencies.sh

Furthermore, the application also requires correctly setting up the database Tcount and table Tweetwordcount in database Tcount, as well as permission configurations for user “postgres”. These requirements are satisfied by **configure_postgres.sh** and running **set_up_db.py**.

Implementation Details
This section deals with the major changes to the original code provided, and its rationale.

Tweets.py
1. **Line 13-16**: populating the consumer key, secret and access_token and access_token_secret to connect to Twitter API for streaming.

Parse.py
1. **Line 9-15**: define a set of “low value” words to use for filtering.
2. **Line 46**: use regexp to filter out all non-alphabetical characters in a word. I noticed that some characters (e.g. the ' in There's) was causing problems to inserting into Postgres SQL and since they may occur anywhere within a string, I used regexp to remove them.
3. **Line 50**: filter out “low value” words. I noticed that some top results without filtering usually goes to words such as “I”, “the”, “you”, “a”, “this”, which does not provide any extra information to detecting trending subjects.

Wordcount.py
1. **Line 13-15**: on initialization, always query the database and get the existing word count.

Since the application may be terminated and restarted by the user, this is to ensure that the application builds on top of what's already in the database and get continuous results.

2. **Line 33-37**: on processing, if the word already exists in the database, then simply update the count, otherwise create a new entry in the database.

Histogram.py

1. **Line 7-18**: I noticed that Python sys module parses output slightly differently for different ways the two numbers were given (e.g. histogram.py 2, 3 | histogram.py 2 3 | histogram.py 2 , 3 | histogram.py 2 ,3). The code here is to deal with any format of input.

Sample Results

Sample output are stored in /exercise_2/screenshots

File	**Description**
Plot.png	A bar chart of top 20 word count, the original implementation (i.e. *without* "low value" words filtering in parse.py)
Plot_Revised_Model.png	A bar chart of top 20 word count, the revised implementation (i.e. *with* "low value" words filtering in parse.py)
screenshot_finalresults_EC2.png	Results returned by **finalresults.py** in Amazon EC2
screenshot_finalresults_no_argument.png	Results returned by **finalresults.py** in the author's Linux box
screenshot_finalresults_with_argument.png	Results returned by **finalresults.py** and supplying an argument in author's Linux box
screenshot_histogram.png	Results returned by **histogram.py** in author's local Linux box
screenshot_histogram_EC2.png	Results returned by **histogram.py** in Amazon EC2
screenshot_storm_topology.png	Topology of the streaming application
screenshot_twitterStream_EC2.png	Results of the streaming application running in Amazon EC2
screenshot_twitterStream_original.png	Results of the streaming application (original implementation) running in author's Linux box
screenshot_twitterStream_revised.png	Results of the streaming application (revised implementation) running in author's Linux box

Potential Improvements

Here we briefly discuss the areas of improvement:

1. Storing the API keys in **tweet.py** and tokens in the source code generally tends to be poor style and exposes to security risk. A better way to do it is to read from a file and it is up to the user to get the file (i.e. potentially using his own credential);
2. Storing a long set of words (in **parse.py**) that appear uninteresting works, but a better way to do it is to use some sort of common dictionary. For example, if we are interested in trending subjects from Twitter feed, then maybe focusing on only nouns, verbs and special words is more interesting than looking at everything.