CSE 435/535 Information Retrieval

Project One : Data ingestion and Solr setup

Due Date : **19th September 2016, 23:59 EST/EDT**

Version 1.1 (9/12/16)

1. Introduction

The primary purpose of this project is introduce students to the different technical aspects involved in this course and subsequent projects. By the end of this project, a student would have achieved the following:

- Setup an AWS account and simple EC2 instances
- Learn about the Twitter API, and querying twitter using keywords, language filters and geographical bounding boxes
- Setup a Solr (a fully functionality, text search engine in Java) instance - understand basic Solr terminology and concepts
- Index thousands of tweets in multiple languages
- Setup a quick and dirty search website showcasing their collected data and implementing faceted search..

The specific challenges in completing this project are as given below:

- Figure out specific query terms, hashtags, filters etc to use in order to satisfy the data querying requirements.
- Correctly setup the Solr instance to accommodate language and Twitter specific requirements

The rest of this document will guide you through the necessary setup, introduce key technical elements and then finally describe the requirements in completing the project. This is an individual project and all deliverables MUST be submitted by 19th September 23:59 EST/EDt

2. Pre-requisites

We would be using Amazon AWS for all projects in this course. Before we begin, you would thus need to sign up for an AWS account if you don't already have one : https://aws.amazon.com. Although the sign-up requires a credit card for verification purposes, you can use a gift card instead. Note that you can even share a gift card amongst a group if you desire. We do not anticipate students using more than their free tier allocation.

UB is a part of the AWS educate program. It gives you $100 in annual credit. Follow instructions at https://aws.amazon.com/education/awseducate/ to claim after you have signed up for your AWS account.

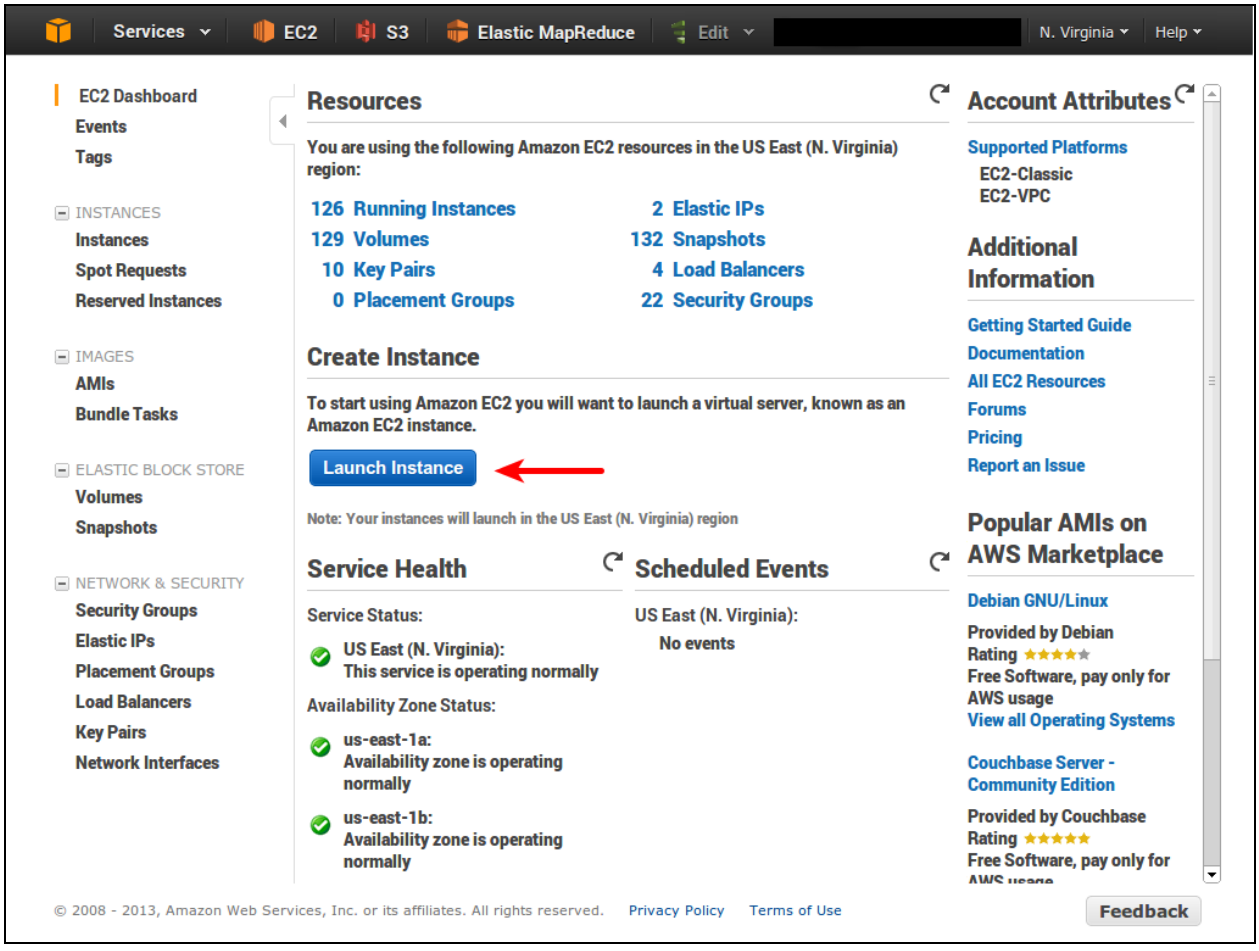
You will also need a Twitter account to be able to use the Twitter API for querying.

3. Setup

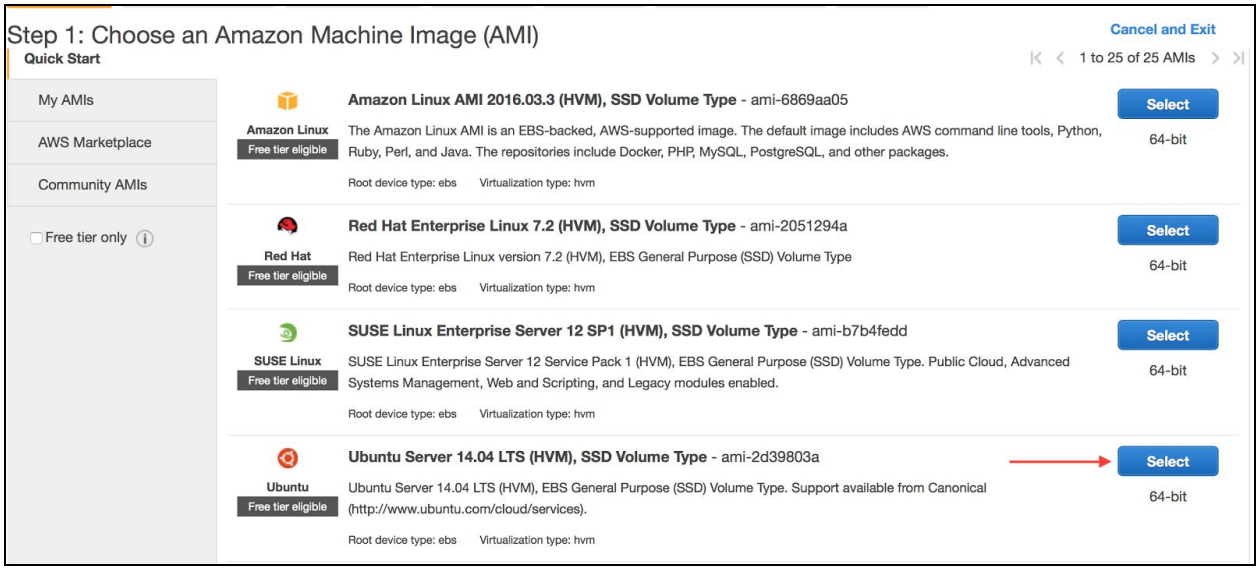
3.1. EC2 setup

Although there are several guides available, instructions here are adapted from Solr's EC2 guide here : https://wiki.apache.org/solr/SolrOnAmazonEC2

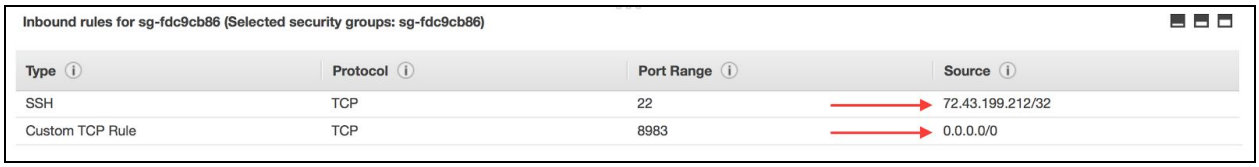
1. Login to your AWS account and navigate to the EC2 dashboard.
2. Create an instance
    a. Click on "Launch Instance"



    b. Select an AMI type. For this demo, we are using Ubuntu 14.04 LTS

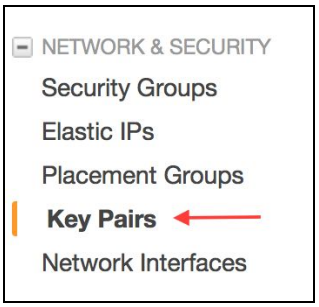


c. Choose an instance type. We keep the default option (General purpose, t2.micro). You may need to change this to t2.small for your project later.
d. Keep the default options for steps 3,4 and 5 (Configure Instance, Add Storage and Tag Instance)
e. Create a new security group. Provide access to SSH for your IP and global access for port 8983. We will later provide a more restricted IP list. You could restrict it to your IP for the time being.



f. Review and Launch!

3. Create a keypair to log in to your instance.
   a. Click on “Key Pairs” under the “Network and Security” group in the left pane



   b. Create a new key pair by giving it some meaningful name. Download and save the file. For most Unix/Linux based systems ~/.ssh is a good place. However, make sure that the security permissions on the folder are set to just you.
4. Login and verify.

a. By now your instance must be up and running. Find its hostname or ip address.
b. Login using ssh as : `ssh -i ~/.ssh/[KEYNAME.pem] ubuntu@[HOSTNAME]`

3.2. Solr setup

It is fairly easy to have a Solr instance up and running, at least for sanity checks.

1. Choose some location where you would install Solr. Say ~/solr.
2. Navigate to that directory and download solr : `curl -O` http://www.gtlib.gatech.edu/pub/apache/lucene/solr/6.2.0/solr-6.2.0.tgz
3. Untar : `tar xf solr-6.2.0.tgz`
4. Install Java 8

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

5. Start a standalone server : `bin/solr start -p 8983 -e techproducts`
6. Verify the instance works. Open http://<ip address:8983>/solr in browser.
7. Index data using : `bin/post -c techproducts example/exampledocs/*.xml`
8. Verify data is indexed.

4. Collecting data

There are three main elements that you need to know with regards to using the Twitter API : Authentication, Streaming vs REST APIs and Twitter Clients.

4.1. Authentication

Twitter uses OAuth to authenticate users and their requests to the available HTTP services. The full OAuth documentation is long and exhaustive, so we only present the relevant details here.

For the purpose of this project, we only wish to use our respective Twitter accounts to query public streams (more about that in the following section). The suggested authentication mechanism for this use case is generating tokens from dev.twitter.com as follows: (adapted from https://dev.twitter.com/oauth/overview/application-owner-access-tokens)

- Login to your Twitter account (create one if haven't already done so)
- Navigate to apps.twitter.com
- Click on "Create New App" on the upper right corner.

- Fill in all required fields. The actual values do not really matter but filling some meaningful values is recommended.
- Once created, within the application details, you would get an option to “Create my access token”
- Click on the link and generate your access token.
- At the end of this step, you should have values for the following four fields under the “Keys and Access Tokens” tab : Consumer Key, Consumer Secret, Access Token and Access Token Secret. You will need these four values to be able to connect to Twitter using a client and querying for data.

4.2. Streaming and REST APIs

We are only concerned about querying for tweets i.e. we do not intend to post tweets or perform any other actions. To this end, Twitter provides two types of APIs : REST (which mimics search) and Streaming (that serves “Live” data).

You are encouraged to experiment with both to see which one suits your needs better. You may also need a case by case strategy - search would give you access to older data and may be more useful in case sufficient volumes don’t exist at a given time instant. On the other hand, the Streaming API would quickly give you thousands of tweets within a few minutes if such volumes exist. Both APIs return a JSON response and thus, you would need to get yourself familiarized with the different fields in the response.

Please read up on the query syntax and other details here : https://dev.twitter.com/rest/public/search. You may be interested in reading up on how tweets can be filtered based on language and/or geolocation. These may help you in satisfying your language requirements fairly quickly.

Similarly, documentation for the Streaming API is present here : https://dev.twitter.com/streaming/overview/request-parameters. Since we are not worried about exact dates (but only ranges), either of the APIs or a combination may be used. We leave it to your discretion as to how you utilize the APIs.

4.3. Twitter Clients

Finally, there is a plethora of Twitter libraries available that you can use. A substantial (though potentially incomplete) list is present here : https://dev.twitter.com/overview/api/twitter-libraries. You are welcome to use any library based on your comfort level with the library and/or the language used.

5. Indexing

Before we describe the indexing process, we introduce some terminology.

5.1. Solr terminology

- Solr indexes every **document** subject to an underlying **schema**.
- A schema, much akin to a database schema, defines how a document must be interpreted.
- Every document is just a collection of **fields**.
- Each field has an assigned primitive (data) **type** - int, long, String, etc.
- Every field undergoes one of three possible operations : *analysis*, *index* or *query*
- The analysis defines how the field is broken down into tokens, which tokens are retained and which ones are dropped, how tokens are transformed, etc.
- Both indexing and querying at a low level are determined by how the field is analyzed.

Thus, the crucial element is configuring the schema to correctly index the collected tweets as per the project requirements. Every field is mapped to a type and each type is bound to a specific tokenizer, analyzer and filters. The schema.xml is responsible for defining the full schema including all fields, their types and analyzing, indexing directives.

Although a full description of each analyzer, tokenizer and filter is out of the scope of this document, a great starting point is at the following wiki page ; https://cwiki.apache.org/confluence/display/solr/Understanding+Analyzers,+Tokenizers,+and+Filters. You are encouraged to start either in a schemaless mode or start with the default schema, experiment with different filters and work your way from there.

5.2. Indexing strategies

This is the part where students need to figure out the appropriate way to index their collected tweets. Overall, there are two overarching strategies that you must consider:

- Using out-of-the-box components and configure them correctly. For example, the StopFilter can be used to filter out stopwords as specified by a file listed in the schema. Thus, at the very minimum, you would be required to find language specific stopword lists and configure the filters for corresponding type fields to omit these stopwords.
- Pre-processing tweets before indexing to extract the needed fields. For example, you could preprocess the tweets to extract all hashtags as separate fields. Here again, it is left to your choice of programming language and/or libraries to perform this task. You are not required to submit this code.

Solr supports a variety of data formats for importing data (xml, json, csv, etc). You would thus need to transform your queried tweets into one of the supported formats and POST this data to Solr to index.

6. Project requirements

We now describe the actual project. As mentioned before, the main purpose of this project is to index a reasonable volume of tweets and perform rudimentary data analysis on the collected data. We are specifically interested in tweets on the following topics:

- US Presidential elections (Politics)
- Syrian Civil War (World News)
- US Open - Tennis (Sports)
- September Apple event, iPhone 7, Watch 2 etc (Tech)
- Game of Thrones (Entertainment)

Apart from English, you should collect tweets in the following languages:

- Spanish
- Turkish
- Korean

The above topics are intentionally specified in a broad sense and this brings us to the first task you need to perform.

Task 1 : Figure out the required set of query terms, language filters, geolocation filters and combinations thereof to crawl and index tweets subject to the following requirements:

1. At least 50,000 tweets in total with not more than 15% being retweets.
2. At least 10,000 tweets per topic.
3. At least 5,000 tweets per language other than English, i.e Spanish, Turkish and Korean
4. At least 5,000 tweets collected per day spread over at least five days, i.e., for the collected data, the tweet dates must have at least five distinct values and for each such day there must be at least 5,000 tweets. Essentially, you cannot collect say 20,000 tweets on one day and split the rest between other four days.

Note that the above are the minimum requirements. You are free to crawl tweets in other languages outside this list (or crawl tweets without a language filter for example) as long as the above requirements are met. Further, this data would be validated against your Solr index. Thus, based on how you setup your indexing, you may lose some tweets and/or have duplicates that may reduce your indexed volumes. Hence, it is encouraged that you accommodate some buffer during the crawling stage.

Once you have collected your tweets, you would be required to index them in a Solr instance. You would need to tweak your indexing to adhere to two distinct sets of requirements - language specific and Twitter specific as described below. Please see the section on Grading for some sample queries that your system must be able to handle.

Task 2 : Index the collected tweets subject to the following requirements:

1. Underlying tweet topic : one amongst politics, news, entertainment, sports and tech.
2. One copy of the tweet text that retains all content (see below) irrespective of the language. This field should be set as the default field while searching.
3. Language of the tweet (as identified by Twitter) and a language specific copy of the tweet text that removes all stopwords (language specific), punctuation, emoticons, emojis, kaomojis, hashtags, mentions, URLs and other Twitter discourse tokens. Thus, you would have at least five separate fields that index the tweet text, the general field above plus four for each language. For any given tweet, only two of the five fields would have a value.
4. Separate fields that index : hashtags, mentions, URLs, emoticons+ (emoticons + emojis + kaomojis)
5. Additionally index date, geolocation (if present), and any other fields you may like.

As a final requirement, you would be hosting a simple web application that connects to your Solr instance and allows you to test and validate your indexed data.

7. Submitting your project

We would use the CSE submit script. You would be required to simply submit the URL of your EC2 instance in a text file as part of your submissions. Some naming conventions are required to be used as described below:

1. Name your submission as project1.txt. It should only contain the IP address of your EC2 instance.
2. Run your solr instances on port 8984. Make sure that the port is accessible.
3. Name your Solr instance as IRF16P1. So if the IP address of your EC2 instance is aa.bb.cc.dd, the Solr query page should be accessible as http://aa.bb.cc.dd:8984/solr/#/IRF16P1/query
4. Name your web application landing page as search.html. So the website URL becomes http://aa.bb.cc.dd:8984/solr/search.html
5. The field names are as given below
    - topic : One of the five topics
    - tweet_text : Default field
    - tweet_lang : Language of the tweet from Twitter as a two letter code.
    - text_xx : For language specific fields where xx is at least one amongst en (English), es (Spanish), tr (Turkish) and ko (Korean)

- hashtags, mentions, tweet_urls and tweet_emoticons for the respective self explanatory values
- tweet_date : Date of the tweet, rounded to the nearest hour and in GMT
- tweet_loc : Geolocation of the tweet.

To submit your project, from any CSE server run submit_435 or submit_535 based on your enrollment (435 for undergrads, 535 for grads).

8. Grading

This project is worth a total of 10 points, these are distributed as follows:

Task	Criterion	Description	Points
Task 1	Tweet volumes	Validate at least 50,000 tweets + at least 10,000 tweets per topic	1
	Language volumes	Validate language volumes - at least 5,000 tweets for tr, ko and es	0.5
	Date criterion	Validate at least five days with at least 5,000 tweets	1
	Retweet counts	Validate at most 15% retweets	0.5
Task 2	Sanity	Validate Solr instance runs + can run some queries	2
	Schema validation	All fields are named as required, contain values as required etc.	2
	Topic adherence	Analysis of top K terms by topic, language and date	1.5
	Data sanity	Analysis based on top K hashtags, URLs, mentions and emoticons	1.5

We will run the grading script once before the deadline (date TBD), allowing you to perform a sanity check of your submission. The final grading script will be triggered right at midnight and thus any late submissions will not be considered.

Appendix

Character encoding

One of the first issues you might encounter or have to deal with is character encoding. Every written character is encoded using one or the other character sets. Most programs (including your browser, favorite text editor, etc.) use some default encoding that is usually determined by your operating system. While using English or most Latin derived languages, the ASCII character set is sufficient and does not pose major problems. However, most other languages we have chosen for this project use extended character sets. For most scenarios, UTF-8 encoding might suffice.

However, there is the issue of UTF-16 encoding and Unicode characters. Some languages like Chinese and Korean, require two bytes to store one character (as against the usual norm of one byte per character). These languages thus,sometimes require UTF-16 encoding that allows storing these extended character sets.

Unicode is an industry standard that supports about 128,000 different characters. It is split into different code point ranges and each each range is mapped to a character range. We describe the relevant code points for emojis in the following section. However, we mention Unicode here to make a point that every character (using any character set) is mapped to a unique unicode code. So you may encounter the terms unicode, UTF-8 and UTF-16 interchangeably in relevant documentation and should learn more to understand the nuances.

Finally, in case you start seeing garbled characters when you try and read your tweets, check your encoding!

Emoticons, Emojis and Kaomoji

Even though they are used for similar purposes to express different emotions; emoticons, emojis and kaomojis use different character sets.

Emoticons

Emoticons are predominantly represented using punctuation, letters and numbers. Thus, in most scenarios the ASCII character set is sufficient to express all emoticons. However, a given emoticon may have several variants (:), :-), :-] are all smiley faces for exampy, le). Further, the overloaded usage of common characters makes it hard to programmatically distinguish between punctuation usage and emoticons. For example, a regular expression trying to match contiguous punctuation would match both ‘!!!!’ and ‘:)’. Using a curated lexicon may provide a decent amount of precision but may suffer in terms of recall.

Kaomojis

There is an alternate “Japanese” style of emoticons (`(-_-;)` and (ﾉ°□°）ﾉ ︵ ┻━┻ for example) that does not require tilting one’s head to understand the emoticon. They use extended character sets, may or may not be enclosed within parenthesis and again, could have multiple variants. Culturally, they may be more frequent in some languages (Korean, Japanese) than others (Turkish, English).

Emojis

Emojis like □, □ and □ on the other hand are more expressive ideograms that instead use distinct character sets. Unicode 9.0 reserves 1,123 characters spread across 22 code blocks as emojis. Recently, applicable emojis may be annotated with Fitzpatrick modifiers to indicate diversity (□ to □□for example). One of the decisions that you may have to make is to figure out if you would use the Fitzpatrick modifiers in your indexing process (i.e preserve the modifications) or ignore them completely and only store the default emoji.

Emojis may appear differently between different operating systems. However, all of them map to the same underlying unicode character. Thus, although emojis have fixed unicode ranges, they are slightly more challenging to handle programmatically. You are encouraged to look at specific examples of handling emojis in the programming language you intend to use.