Table of Contents

[Setting Up Eclipse – With STIRS Code & Jars 2](#_Toc394497636)

[Acquiring STIRS Code 2](#_Toc394497637)

[The API 2](#_Toc394497638)

[Using the API 3](#_Toc394497639)

[Setting Up Twittertools/Maven 3](#_Toc394497640)

[How STIRS uses the API 4](#_Toc394497641)

[Using GitHub 4](#_Toc394497642)

[Using STIRS Account/Organization 4](#_Toc394497643)

[Creating Repositories & Adding Code 5](#_Toc394497644)

[Forking STIRS & Adding Code 7](#_Toc394497645)

[Running STIRS 9](#_Toc394497646)

[Arguments 9](#_Toc394497647)

[Projects’ Information 13](#_Toc394497648)

[API 14](#_Toc394497649)

[LinkCrawling 16](#_Toc394497650)

[Machine\_Learning 17](#_Toc394497651)

[Main 18](#_Toc394497652)

[Miscellaneous 18](#_Toc394497653)

[nistEvaluation 19](#_Toc394497654)

[Google 19](#_Toc394497655)

[Wordnet 20](#_Toc394497656)

[Query\_ExpansionInternal 21](#_Toc394497657)

[Relevance 21](#_Toc394497658)

[TTG 22](#_Toc394497659)

[Twitter-Tools-Core 22](#_Toc394497660)

[Useful Links 22](#_Toc394497661)

# Setting Up Eclipse – With STIRS Code & Jars

Hopefully, by the time you’ve started working in STIRS, you know how to use Eclipse. If not, here (<http://www.tutorialspoint.com/eclipse/eclipse_overview.htm>) is a pretty good tutorial that will give you a rundown on how to install and use the basic features of Eclipse. This guide is going to expect that you have Eclipse installed and know how to use the basics.

## Acquiring STIRS Code

At the time of writing this guide, Dr. Small and Dr. Lim were both faculty mentors to the project and given .zip files of the 2014 & 2012 code – these include all the .java, .txt and .jar files you’ll need. You’ll be using GitHub (hopefully – it’s assumed in this guide) to sync your code up with other group members, so please refer to our GitHub Guide, written later in this guide, to figure out how to set up Git on your computers.

If you wish to just use the .zip file, feel free to download the 2014 code (2012 is for reference only), unzip it to your preferred directory, and import into Eclipse. Otherwise, the GitHub Guide goes into detail about downloading it the Git way. Note: If you choose to do the .zip file, you’ll have trouble later on using Git (unless you create a new repository with the code), so I suggest you go ahead and download it using GitHub.

All jar files can be found in the libraries folder of both the GitHub repository and the .zip file. A list of jar files used:

* apache-mime4j-0.6.jar
* commons-codex-1.3.jar
* common-logging-1.1.1.jar
* httpclient-4.0.jar
* httpcore-4.0.1.jar
* httpmime-4.0.jar
* jaws-bin.jar
* jdom-1.1.jar
* jericho-html-3.2.jar
* jsonic-1.2.0.jar
* jsoup-1.6.0.jar
* junit-4.7.jar
* jwnl-1.3.3.jar
* langdetect.jar
* libthrift-r771587.jar
* log4j-1.2.14.jar
* lucene-core-3.2.0.jar
* myjericho.jar
* opennlp-maxent-3.0.3.jar
* opennlp-tools-1.5.3.jar
* opennlp-uima-1.5.3.jar
* pdfbox.jar
* weka-src.jar
* weka.jar

# The API

In 2011 & 2012, the last two years that STIRS participated in the Microblog Track, the corpus was a downloadable tweet corpus, where you had access to all tweets and could manipulate it as you with. In 2013, they introduced an API, where you could “query” (contact) the API server and request a number of tweets based on the topic.

In this section, we’ll be explaining how to use the API through a basic terminal window, how to set up Twittertools & Maven in Eclipse, and how Twittertools interacts with our code.

## Using the API

If you want to see it work from a terminal, check out the API specifications here: <https://github.com/lintool/twitter-tools/wiki/TREC-2013-API-Specifications>, under “Using the Simple Java Client”.

In 2014: The hostname is: nest.umiacs.umd.edu. The token is: ddf20f. The group is: tw13t016.

When using them with the STIRS code, they should be modified in the API\_Query.java class.

## Setting Up Twittertools/Maven

**NOTE:** These instructions assume a version of JAVA is already installed.

1. Install Maven: Install maven for your operating system using the instructions at <http://maven.apache.org/download.cgi#Installation>

**NOTE:** The above instructions include proper configuration of the JAVA\_HOME and M2\_HOME environment variables on your machine. Be sure to include the correct paths to your JDK for JAVA\_HOME and your maven repository for M2\_HOME as they are very important to successfully configuring the API.

1. **OPTIONAL:** Install git for Linux. Open an instance of xTerm and enter:

* sudo apt-get git
* Enter password
* When prompted about memory space, hit Y

Clone twitter-tools: Clone twitter-tools from github using the command line with git installed. Follow these instructions, under the heading “Getting Started”: <https://github.com/lintool/twitter-tools>

**ALTERNATIVE:** Navigate to <https://github.com/lintool/twitter-tools> and manually download the package through your browser by selecting “Clone to Desktop” or “Download ZIP” under the clone URL.

1. Generate a Maven build: Build Twitter-tools using maven. The commands for this are also located at <https://github.com/lintool/twitter-tools>. For use with Eclipse, both build configurations (regular build and eclipse:eclipse build) are necessary.
2. Install the Maven plugin for eclipse: Install the Maven plugin for eclipse using the instructions at <http://www.eclipse.org/m2e/>. You can do this many different ways. Two of the easiest ways are below:
3. Download an Eclipse distribution with the Maven plugin included.
4. Access the Eclipse marketplace via your current eclipse version and find the M2E plugin available for download.
5. Import twitter-tools to eclipse: Import twitter-tools into Eclipse using the maven plugin by right clicking the package explorer, selecting import -> Maven -> Existing Maven Projects, then browsing to the twitter-tools-core directory created by step 3. The plugin will ask you to import 2 projects, /twitter-tools-core/ and /twitter-tools-RM3/, you only need twitter-tools-core.
6. Update archive libraries: TREC 2014 participants found that after importing the maven build of twitter-tools, we were still missing some archives. Eclipse will tell you which JAR files you still need, if any. Search google or a repository site for the missing files and download them. Then, add them to your build path by right clicking the project in the package explorer, selecting Build Path-> Configure Build Path -> Libraries Tab -> Add External JARS. It is wise to keep any JARs necessary to the build path in one folder, we called it Libraries.

## How STIRS uses the API

Because most of our code is used to using a tweet index to find tweets, we organized the API to download each topic’s tweets into a permanent index (which can be found at TREC2014 🡪 src 🡪 API 🡪 index). Each .txt file is in the format of “MB001.txt”, with “MB001” representing the topic it was downloaded for. These index files are not changed during runs, and should only be updated when downloading a new collection with new topics.

In the main Stirs.java class, you’ll find a method called “apiQuery”, which uses the API\_Query class to contact the API and gather each topic’s tweets. This method is only called when updating the topics (see *Running STIRS, Arguments* for how to force the system to update the API).

For more information on how Stirs, API\_Query and the rest of the classes work, look at *Running STIRS, Projects’ Information*.

# Using GitHub

GitHub (<https://github.com/>) is a website often used by developers working on the same set of code. This is to help with merging difficulties – when many developers are working on the same set of code, it can sometimes be hard to have multiple copies floating around. GitHub keeps a central repository of the code, and allows members of that repository to fork (create a copy) the repository, so each member has their own copy. When they wish to merge their copy with the central repository, they send in a pull request, and any owner of the central repository has the ability to merge the pull request (a commit) to the main code.

This guide will show you how to use GitHub in relation to STIRS. The first thing you should do is create an account on GitHub and download Git onto your computer (<http://git-scm.com/downloads>).

## Using STIRS Account/Organization

A main STIRS account has been created, that hosts the organization SienasTwitterInformationRetrievalSystem. All members have access to this account, and should be used to 1) add owners to the organization (those group member participating in this year’s track) and 2) merge all requests to the main STIRS code.

Username: STIRS

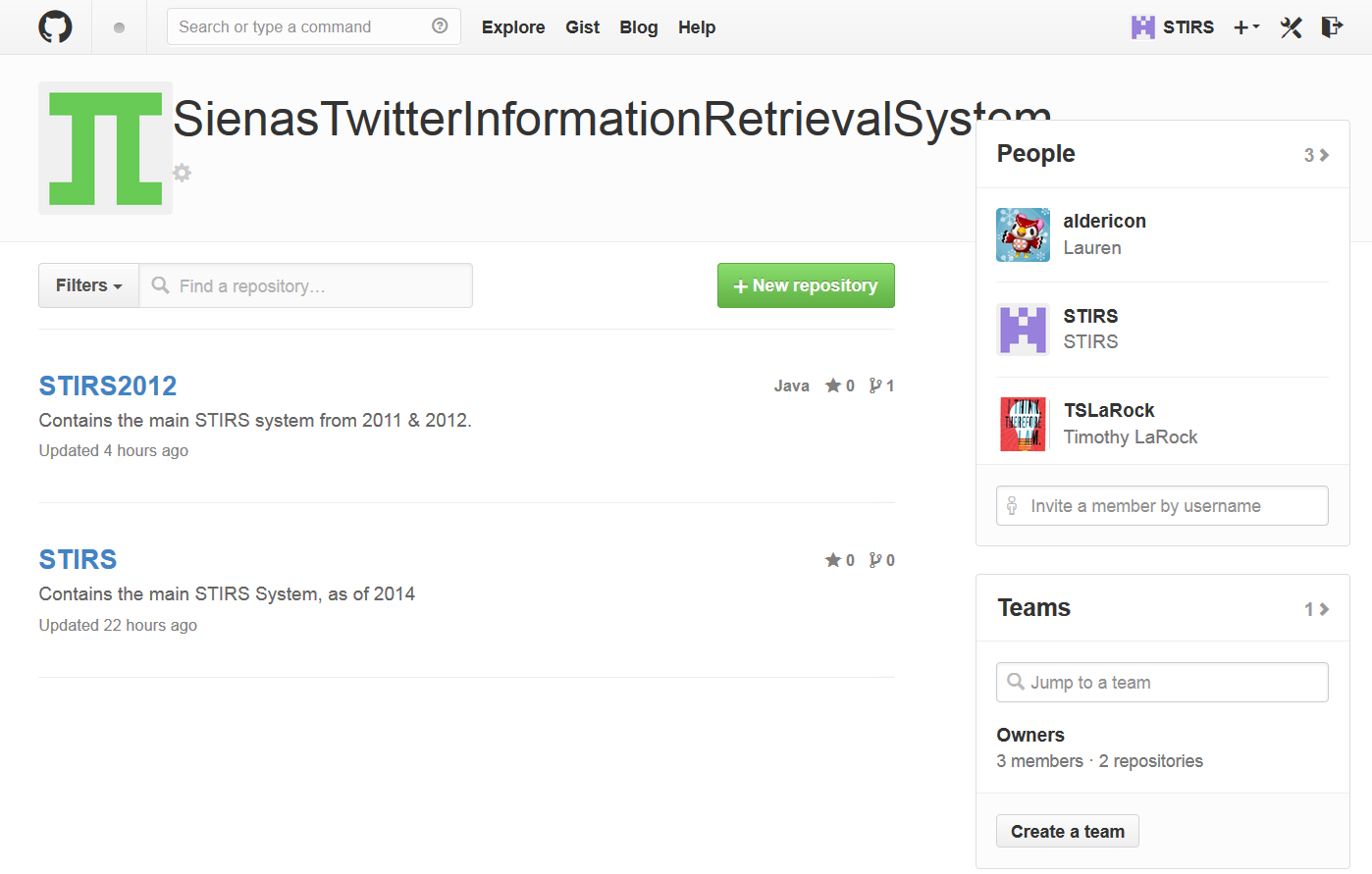
Password: sTiRs2014

Feel free to add your email to the list of emails on this account.

The current owners should remain STIRS, Dr. Lim’s account (DarrenLim4) and Dr. Small’s account. A new group should be created for the year it’s being worked on (ex. STIRS2014).

When logging into this account, please add your account(s) to the group for this year (e.g. STIRS20##) for the organization. The organization can be found under “Organizations” in the STIRS account profile page, and the arrow next to “People” can add people to that organization and to that specific group.

All past years should only have read access – please modify if needed.



## Creating Repositories & Adding Code

As you can see above, the organization has two repositories: STIRS2012, which hosts code created from 2011 & 2012 and is considered deprecated, and STIRS, which contains the main, and current, STIRS code.

I suggest creating a separate repository for each year, while keeping the main repository (STIRS) to your current code. So, if STIRS is used next year in 2015, create a repository called “STIRS2014” that contains the code you were given, but then continue to use STIRS as your current code.

To create a new repository, simply click the green “+New repository” button on the organization’s page. You can call, and describe, it however you’d like, just try to follow good naming conventions. Make sure to click the option to “Initialize this repository with a README”, but make sure the .gitIgnore and the license are both “None”.

Once created, STIRS (the account) should create its own fork of the repository; to fork a repository, go to the repository’s page and click the Fork button:

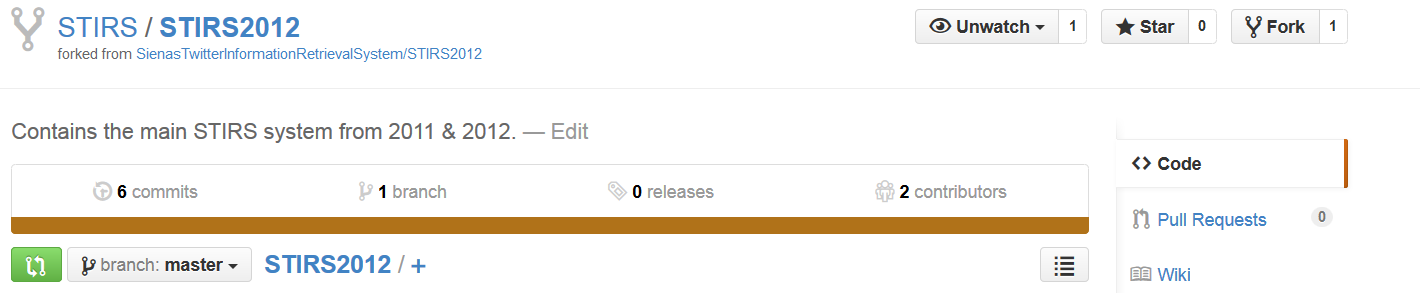


This will create a fork on GitHub of the repository on your account.

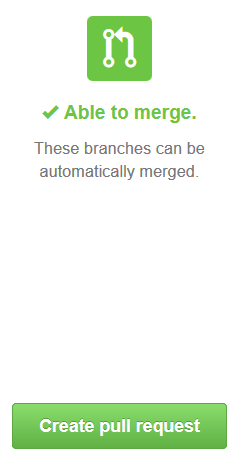
The rest of the setup is done through Git, on your computer. (The following instructions were done on a Windows machine and might have to be modified for your use.)

1. Go to the folder that has the code in it.
2. Right click the folder and select “Git Init Here”.
3. Right click again and click “Git Bash”.
4. Write: git remote add origin https://github.com/USERNAME/REPOSITORY.git (ex. If you forked the repository onto the STIRS account – which should have happened – the username would be STIRS. If the name of the new repository was STIRS2014, then the url would be https://github.com/STIRS/STIRS2014.git)
5. git pull origin master (this will grab the current code in the repository – since it’s a new one, it’ll only have README file. Do NOT do this if the repository isn’t new – go to *Forking STIRS & Adding Code* section instead.)
6. git status (now, this will show you the files that can be committed to the repository. The ones in red – which all files will be read the first time you run git status – are those that aren’t being committed and the ones in green are being committed.)
7. git add FILENAME (go through and add each of these files to the repository. Just be careful and make sure no individual file is more than a couple MBs. GitHub doesn’t like too big files. If the folders do contain big files – none of the java files should this happen to – simply remove it before using git add on the folder name, so it’s not added to the commit.)
8. Once done adding files: git commit –m “DESCRIPTION”
9. git push origin master

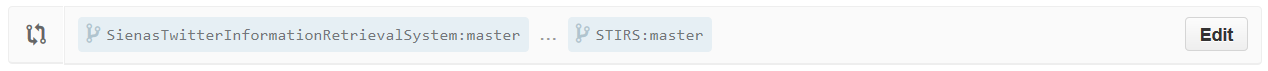
Now your repository will have these files. You can also update README.md in this way. Once done, create a pull request by clicking the green button.



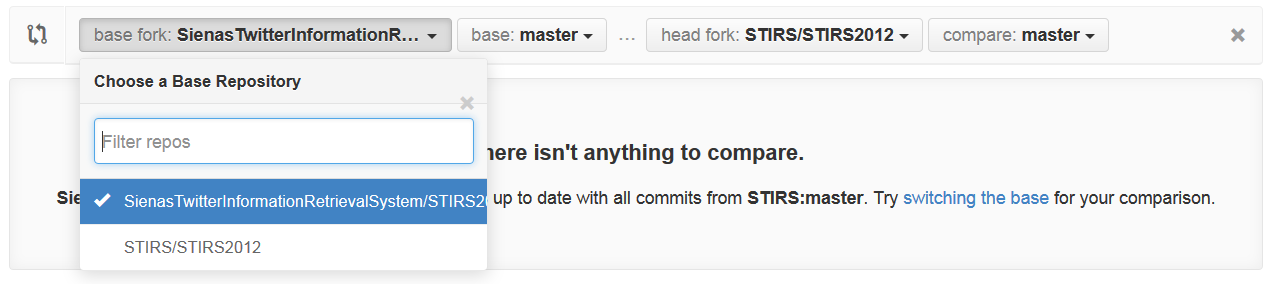
And make sure that you’re able to merge the changes without conflict.



Make sure that the top looks something like this:



In relation to your own fork and repository name – you want to be pushing it to the master version on the organization’s repository from yours. Otherwise edit it to the right one:



Once pushed, you can (and this can be done by any owner for the organization, but try to use the STIRS account) merge the pull request into the main repository and the changes will take effect.

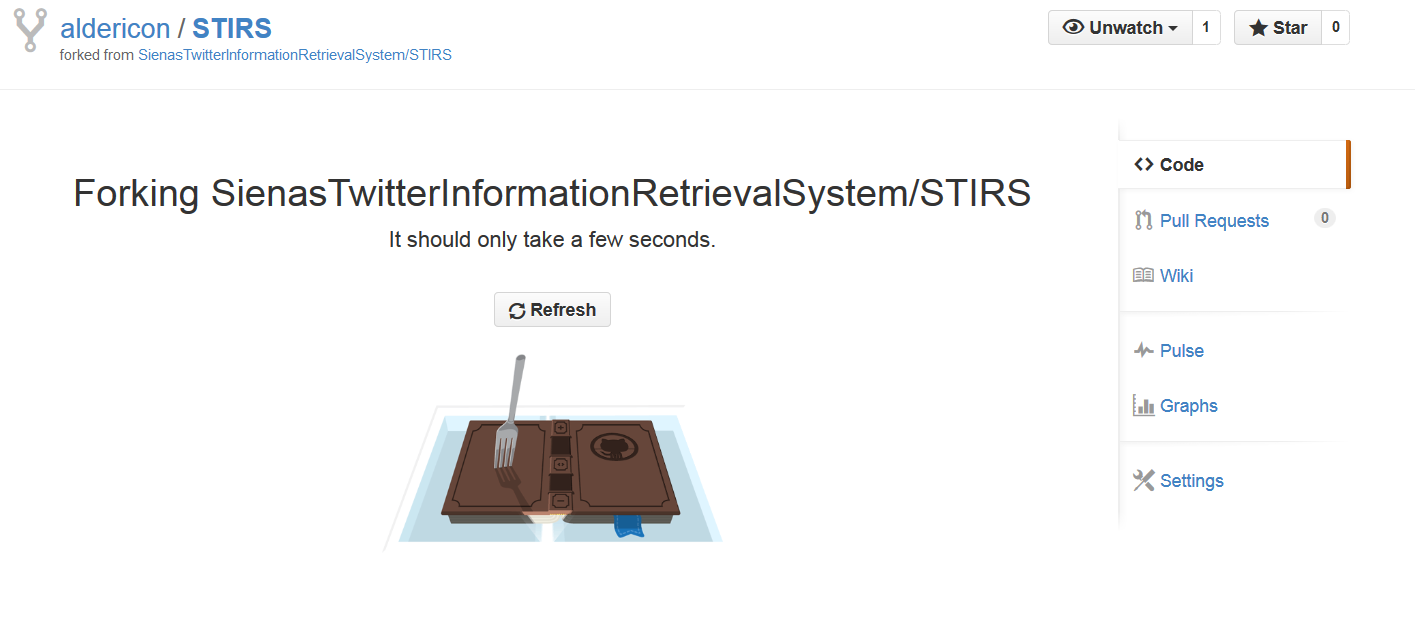
## Forking STIRS & Adding Code

These are instructions for how to fork a current repository that currently has code in it (for creating a new repository, see above section) and then being able to add code to it.

To fork your own repository, click the Fork button next to the repository’s name.



You’ll see something like this pop up:



And VOILA, you have your own copy of the repository on your account. The following instructions were done on a Windows machine and assume you don’t have a copy of the code.

1. Create a permanent directory where the code will be stored.
2. Right click the folder and click “Git Init Here”
3. Right click the folder and click “Git Bash”
4. Write: git remote add origin <https://github.com/USERNAME/REPOSITORY.git>
5. git push origin master

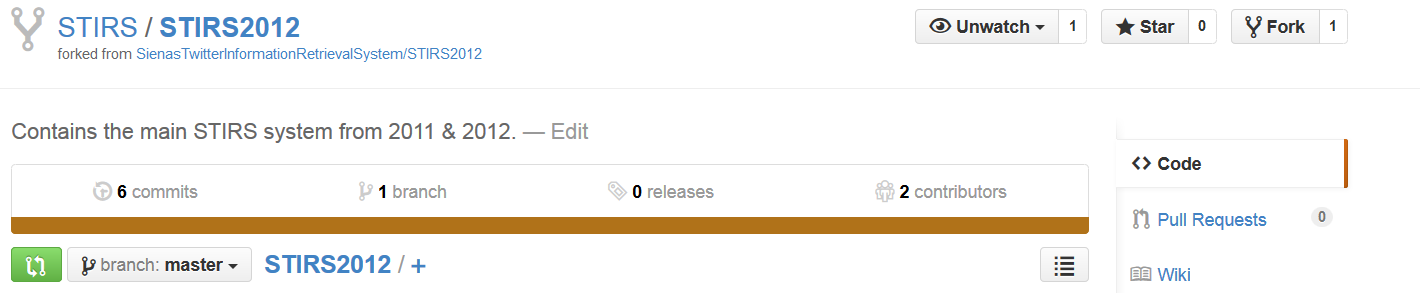
This will create a local copy of all the code on your machine. It can be added and modified in eclipse through the usual method of importing it.

When you want to send code to your forked repository:

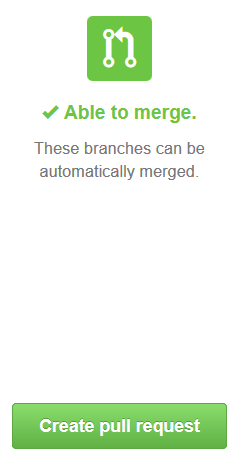
1. Right click the folder and click “Git Bash”
2. git status
3. git add . (change the . to the file name you want to add, if you don’t want to add everything)
4. git commit –m “DESCRIPTION”
5. git push origin master

When you want to merge your changes to the central repository:

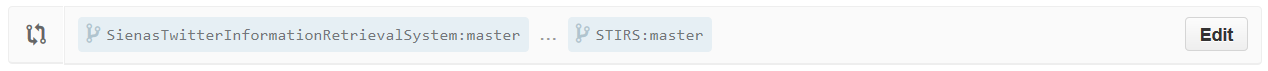
Create a pull request by clicking the green button.



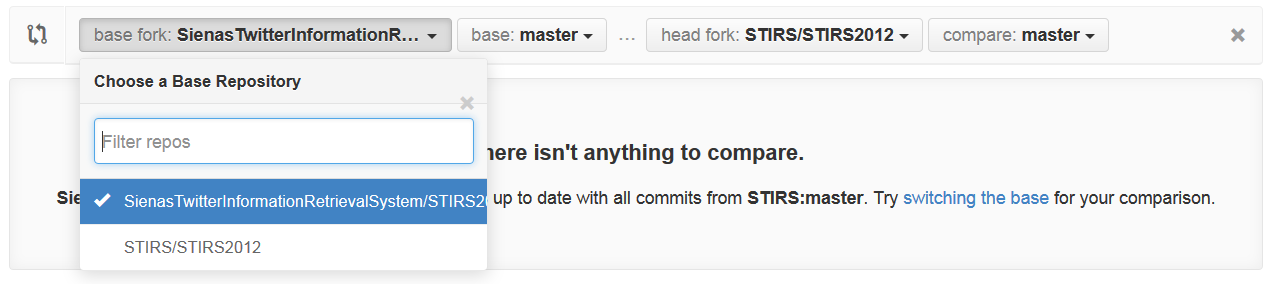
And make sure that you’re able to merge the changes without conflict.



Make sure that the top looks something like this:



In relation to your own fork and repository name – you want to be pushing it to the master version on the organization’s repository from yours. Otherwise edit it to the right one:



Once pushed, you can (and this can be done by any owner for the organization, but try to use the STIRS account) merge the pull request into the main repository and the changes will take effect.

Make sure you tell people you’ve edited the central repository, as they will have to perform “git fetch upstream” 🡪 “git merge upstream” to merge the changes into their local repository on GitHub, and then “git pull origin master” to pull to their local machines.

# Running STIRS

This part assumes you have the code (both TREC2014 and twitter-tools-core) up in the Eclipse folder. At the beginning, there shouldn’t be any errors in the TREC2014 folder. If there are, probably something to do with the Build path – make sure you have all jars installed.

STIRS was made specifically to run for the Microblog Track with the API and the current format of query topics (which are provided with the code). If the API is no longer set up – or is running on a different server – you’ll have to adjust to make sure the code will work.

The following is a guide on how each part of STIRS works and the steps to getting certain modules running.

## Arguments

STIRS was built, especially in 2014, so that everything could run based off of running one program. This “main” class is under TREC2014 🡪 src 🡪 Main 🡪 Stirs.java. If you take a look at the beginning of this program, you’ll see a giant “main” method with a bunch of program parameters – and their descriptions – listed at the top. Each of these program arguments can be turned on or off – or supplied with input – to tell the program what to do.

If you wish to see what a “baseline” run looks like, do the following:

1. Open up Run Configurations
2. Create one, if there isn’t one already, for the main method of Stirs
3. Open up the Arguments tab, and under Program arguments write:
4. –harddrive /home/lmathews/workspace/TREC2014 –q /src/API/topics\_2011.txt –judges /NISTJudgments2011.txt –hits 3000 –tag 2011-2011-baseline –scorecomparator –precision
5. And click Run (make sure to replace the file paths with the correct file path – the only one that should need changing is the –harddrive path) at the bottom.

If everything goes correctly, the file “2011-2011-baseline.txt” inside the folder TREC2014 will contain the precision score (which should also be printed to the screen).

There are a multitude of different arguments to change how the program will print things. Let us go through each of the possible arguments and why they were made; each of these arguments are put to use in the Run Configurations, Program arguments part of running Stirs.java. All are case sensitive.

|  |  |  |
| --- | --- | --- |
| Argument | Description | Required?\* |
| -harddrive | * Supports where the harddrive path is, leading to the project folder. * For the linux machines we used, it was “/home/lmathews/workspace/TREC2014”. But for Windows machines it was “C:\\Users\\Lauren\\workspace\\TREC2014”. * This should be updated whenever the project switches between computers. * This is used to indicate where to find/put files throughout STIRS. * A full list of .java files that need to be updated with their filepaths can be found in the variable “FILEPATHS” in Stirs.java. | Yes |
| -q | * Specifies where to find the topics file * These are listed in TREC2014 🡪 src 🡪 API. We have ones for years 2011-2014. * Should just be starting from src. * Ex: \\src\\API\\topics\_2011.txt or /src/API/topics\_2011.txt * Used to signify what the topics are and what the MB numbers are (as well as information that accompanies each topic). | Yes |
| -hits | * Informs how many tweets to return for the run * When receiving tweets from the API, it’s always the max amount of 10,000. * When updating the links content for Link Crawling module, it’s always set to 10,000. * This should be around 3000 minimum, as you don’t know how many tweets will “make it” for the needed amount of 1000 tweets per topic (TREC requirement for submitting runs). | No |
| -tag | * This signifies what the run will be called. * Only matters when making individual runs which you will need to record results for * Impacts the precision, results, and module files (that are created). * If not indicated, default tag is “default” * For TTG, in 2014, an additional string to the tag was added, “TTGRun”. | No |
| -scoreComparator | * This is used in the Results.java class * Informs whether to sort the tweets by ID, Score or Rank for the run * Usually they want it sorted by rank, which should be the same as sorting by score | No |
| -tm | * Indicates which modules to use for that run * Ex: -tm 1,2,3a,4 * Note: make sure that there are no spaces between the numbers.   Full list: 1 = Link Crawling, 2 = Machine Learning, 3a = Google, 3b = WordNet, 3c = CommonWords, 3d = Links, 3e = HashTags, 4a = rescoreTweets, 4b = Tweet<->Topic Google (Relevance) | No |
| -external | * Originally, it was thought that some anything using outside data – such as slang or a parts of speech tagger – was considered external * We now know it’s more like using external data, like Google or Wikipedia * However, this tag was kept to control whether we wish to turn on certain settings * Used in CommonWords, for controlling “external” input * Used to control cleaning up after query expansion modules are run | No |
| -api | * Whether to force STIRS to resubmit the topics to the API * Otherwise, the program only queries the API if it doesn’t find the files it’s looking for | No |
| -updateContent | * Google Query Expansion involves using the topics to find their links in Google and download the content of those links * Since the topics don’t change between runs, this only needs to be indicated if running this query module for the first time with this collection of topics | No |
| -querySplitting | Whether we wish to use query splitting index for the run. | No |
| -port | What port to use for the API. Currently, there are only two ports:   * 9090: Tweet2011 Collection * 9091: Tweet2013 Collection   2011-2012 used Tweet2011. 2013-2014 used Tweet2013. | No |
| -manual | * Whether the run will include a manual run * Only affects query expansion * Gives the “user” to affect which, of the new queries suggested, to include for the run | No |
| -createTrainingSet | * Affects Machine Learning only * Whether to create a new training set for the collection * Only needs to be run once for a new collection | No |
| -hashtag | * Whether hashtags should be included in CommonWords (Internal Query Expansion) analysis. * Automatically turned on if –tm 3e is turned on | No |
| -TTG | * Whether the run will be used for TTG analysis * Should only be used for the best precision adhoc run | No |
| -useTTG | Whether to use TTG analysis in adhoc task run. Automatically turns –TTG on. | No |
| -judges | * The file name of the NISTJudgments file * Ex: /NISTJudgments2011.txt * All can be found in TREC201 | No |

\*Is this argument required for the program to run every time?

If you want to see where these variables are used, you can highlight them to have them show up through the file, or right click and click “References 🡪 Workspace” to find all places where they are used. This works for class files and methods as well.

One other program that runs by itself is UrlContentRetrieval.java, found under TREC2014 🡪 src 🡪 LinkCrawling. It’s used to create the link content at the beginning of each collection. For more information, see the *LinkCrawling* section.

Before we move on to discussing individual classes, something to note: below Program Arguments in Run Configurations is a “VM Arguments” field. In that field, the following should be put in (for running Stirs.java and UrlContentRetrieval):

-Xms10G

-Xmx11G

-server

-XX:MaxGCPauseMillis=10

Each of these configurations does a different setting. –Xms and –Xmx set the minimum and maximum heap size. This is to keep the program from defaulting to a smaller heap. –server allows the computer to use as many cores as possible, while –XX:MaxGCPauseMillis… “if a pause time goal is specified, the heap size and other garbage collection related parameters are adjusted in an attempt to keep garbage collection pauses shorter than the specified value.” (<http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html#par_gc.ergonomics>)

## Projects’ Information

There are 50 different class files, 16 miscellaneous files and over 1000 index files contained in STIRS. This will go over each part of STIRS and explain what the class is supposed to do, what files are in the package, and if there are any difficult parts of the code in that section.

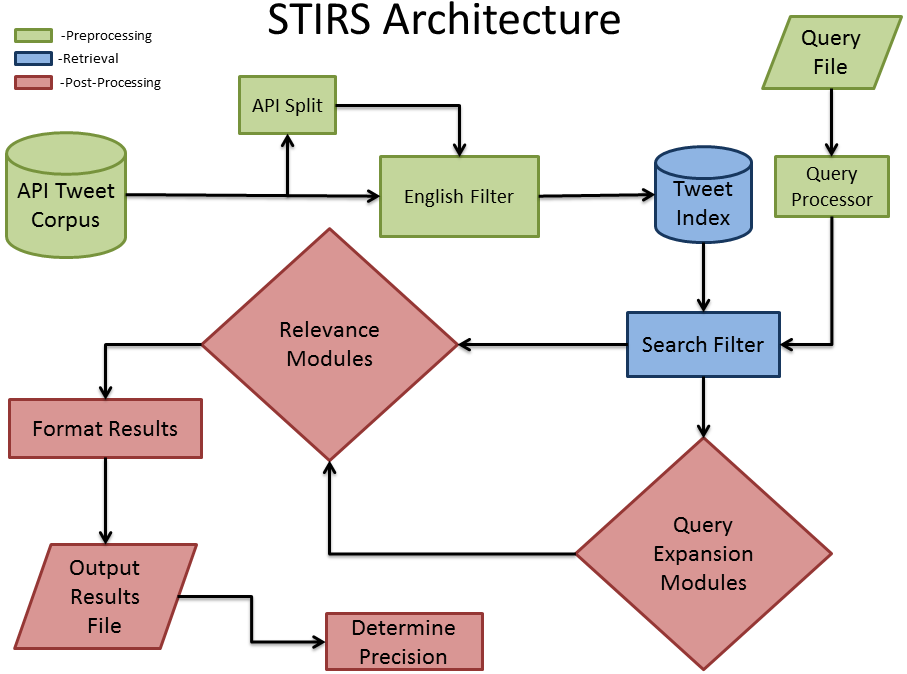


Figure 1: Official STIRS Architecture (created in 2014)

As you can see in Figure 1, this is how the “flow” of STIRS works. The “API Tweet Corpus” is grabbing the tweets from the API and creating the regular and split indexes. Each goes through a filter to remove non-English tweets (automatically ranked as irrelevant by TREC). This creates are own “Tweet Index”, where each topic has its own MB###.txt file. Then, when search for the tweets to be used in the run, it uses the topics file. For each run, it uses multiple relevance and query expansion modules (query expansion always runs first). Then it formats the results and calculates precision, if wanted.

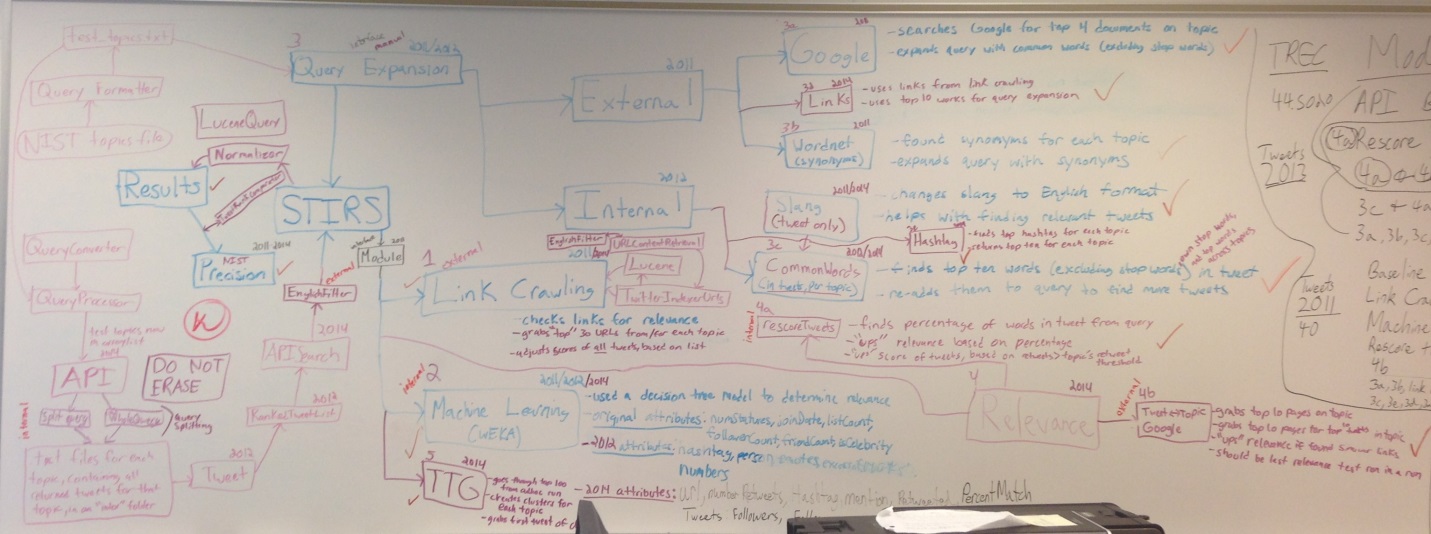


Figure 2: STIRS Modules

Figure 2 (and a large representation of Figure 2 should be included with the documentation) represents STIRS’ modules.

If you wish to know which classes are used, right click the class and click Reference 🡪 Workspace.

Note: Often you’ll find me say that something is a “data structure” class. This means that it makes up a data structure – like an arraylist – where another class uses it to store information. For example, we use the Tweet class to store all the tweet information. Each Tweet contains its own information.

Files found in the TREC2014 Folder:

|  |  |
| --- | --- |
| Name | Description |
| NISTJudgments####.txt | Contains the judgment files, as given by TREC, for each year’s topics. |
| output.csv | Contains each run’s initial tweets and scores. After creating a new collection’s index, contains all tweets found for each topic, so that Link Crawling can find the maximum amount of tweets. |
| output-split.csv | Contains the initial split index tweets, for Link Crawling |
| queExpFile???.txt | Contains the module’s tweets have every run. |
| slangOutput.csv | Contains the new “output.csv”, for CommonWords use only. |
| module???.txt | Outputs the module’s tweets after every run. So module0 contains only the first module, module2 contains results from both first and second modules, etc. |
| TAG.csv | The submission file for the run, if created. |
| TAG.txt | The precision file for the run, if created. |

### API

All API-related files go here. This includes “data structure” classes intended for storing various parts of tweets, formatting the topics and searching/querying the API.

Java Files:

|  |  |
| --- | --- |
| Name | Description |
| API\_Query | * Handles accessing the API code (from twitter-tools-core) * Only called when one needs to update the API Tweet Index (TREC2014 🡪 src 🡪 API 🡪 index OR index-split) * Grabs as many tweets as possible for each topic and gives them a Lucene score/rank   Note: In Stirs.java, in api\_query, there is a “Thread.sleep(5000)” line. Do NOT remove this; keeps us from overloading the API, since other teams need it too. |
| APISearch | * Used for searching through the MB###.txt files created by querying the API (MB###.txt files found at TREC2014 🡪 src 🡪 API 🡪 index OR index-split) * Also used for searching through the Lucene url index (found at TREC2014 🡪 src 🡪 LinkCrawling 🡪URLIndex); it’s the url content downloaded for each tweet (with a url), separated by topic * Creates the tweets that will be used in the run |
| LuceneQuery | * A “data structure” class used to hold the topics * You’ll often find this declaration: ArrayList<LuceneQuery> queries; |
| QueryConverter | Takes in each line of the QueryFormatter’s arraylist and formats it into the LuceneQuery data structure. |
| QueryFormatter | Takes in the queries file, that has the query information on multiple lines, and return an arraylist of size the number of different topics; each part of the arraylist holds a line of all the topic’s information. |
| QueryProcessor | Uses the QueryFormatter and QueryConverter classes to convert the topics into a format useable by our system. This is an arraylist of LuceneQueries, where each part of the arraylist hold information for that topic, which can be gotten by declaring queries.get(i).getQuery(). |
| RankedTweetList | Is a “data structure” class for holding the run’s topic’s tweets. So each RankedTweetList (often shortened throughout the code to rtl) contains an ArrayList<Tweets> for that topic. |
| Tweet | Is a “data structure” class for holding a single tweet. Each tweet has information taken from the API, such as retweet number, followers\_count, the username, etc.  Note: Some parts of this tweet, such as “url”, does not contain the url for the tweet but is used for Link Crawling. |

Files:

|  |  |
| --- | --- |
| Name | Description |
| index | A folder that contains each topic’s (MB###.txt) tweets, collected from the API. This section should not change once collected; only needs to be updated when downloading a new collection. |
| index-split | A folder that contains each topic’s tweets, collected from the API when using query splitting. This section should not change once collected; only needs to be updated when downloading a new collection. |
| index-run | This folder contains each topic’s tweets after they’ve been recollected from the API when all query expansion modules for a run is done. These files are changed every time a run finishes when using a query expansion module, as it must re-query the API to find more tweets related to the new query. |
| q1\_results.txt | Used with query splitting to hold the initial first half of the topic’s tweets. |
| q2\_results.txt | Used with query splitting to hold the initial second half of the topic’s tweets. |
| topics\_####.txt | Each topics folder contains the list of topics given for that year. Each of these also corresponds to a NISTJudgments####.txt file, found in TREC2014. |
| profiles | A folder containing all different language detection, using Google’s language detection (see <https://code.google.com/p/language-detection/> if you need help with that code). Should not be edited. |

### LinkCrawling

This package contains all the classes and files needed for various Link Crawling activities, including running the main “LinkCrawling” module class, gathering the link content and creating an url index.

Java Files:

|  |  |
| --- | --- |
| Name | Description |
| Links | External Query Expansion Module (#3d). Uses the top 30 urls found from Link Crawling the urls from the initial API tweet scrape (both regular and split). For each topic, it finds the top 10 words from the top 30 urls. |
| RankedJoin | Main class for Link Crawling (#1). Grabs top 30 urls from all url content gathered (Lucene does this) for each topic and adjusts each tweet’s score based on:   1. Whether it has a url 2. Whether it made it to the top 30 urls list 3. Whether it has multiple urls |
| TwitterIndexerUrls | Uses Lucene to go through contentFile####.txt and create an index (URLIndex) by topic; each MB folder contains all the url content for that topic. Then, when called, returns the top 30 urls for each topic. |
| UrlContentRetrieval | The only other “stand alone” (works by itself) class besides Stirs.java. Used when first downloading a collection to grab its urls. This should be done right after the collection is downloaded into the system. Each time, a content file with the year (ex. contentFile2011.txt) will be created, so you don’t have to do it more than once for each collection, regardless of different topics.  Note: When link crawling, you’ll sometimes find that it will seem to “hang” (like an infinite loop) on a url; if it still hangs after 20 minutes, you’ll have to stop the program, copy the file to a safe place, set indexStart and urlsProcessed to the last url number + 1, and set urlsCanUse to the last number printed. Start it up; this will cause it to continue and disregard the “bad” url. After it’s done, all you have to do is combine the content files. |
| URLListGen | Used to go through output.csv (which, right after creating the collection, will contain all tweets downloaded – see createTweets() method in Stirs.java) and output-split.csv and create a list of all the urls.  Also has a number of useful methods for finding, removing and returning urls in a tweet. |

Files:

|  |  |
| --- | --- |
| Name | Description |
| URLIndex | For each topic, contains the url content downloaded. It’s separated by topic so that, when gathering the top 30 urls for each topic, it doesn’t grab those urls that aren’t part of that topic. |
| contentFile####.txt | A huge content file, for each year, that contains all the content gathered from the initial scrape of the API. The API should not be recreated once LinkCrawling is done, since it will require the link crawling to be done again. This might take as long as 2 days.  This file will NOT be on GitHub; it’s too big. Get it from the .zip file and do not commit it to GitHub. |

### Machine\_Learning

This package contains all the stuff needed for the Machine Learning module.

Java Files:

|  |  |
| --- | --- |
| Name | Description |
| ARFFCreator | Weka needs to file type “.arff” to create a model. That converts each .csv file into the .arff format for Weka. |
| ProcessAttributes | For each tweet, it finds all the attributes based off the tweet (including relevance). |
| QTweet | A “data structure” class used only by PrcessAttributes when figuring out whether a tweet is relevant. |
| TM2 | The “main” class for Machine Learning (#2). First time through, creates a training set for the collection. When finding relevance, creates a .csv file for each topic and sends it through the model (which is based off the training set). Finds a score and adjusts the tweet score.  Note: The code “Classifier cls = new JRip()” can be edited to reflect different models. I commented those out that worked at the top of the class – feel free to experiment with any of them. More information can be found here: <http://weka.wikispaces.com/Serialization>.  Note2: A new training set can be created for each year, but remember – it relies on the judgment file to be created! Unless you want to manually judge them, use another year’s training set, it won’t be a problem. |
| Weka | Handles finding the score for each tweet based off the model (which is based off the training set) and the tweetSet.csv file given. Called from TM2. |
| WekaTweet | A “data structure” class for holding each tweet’s attribute information. |

Files:

|  |  |
| --- | --- |
| Name | Description |
| trainingSet.arff | The converted trainingSet.csv file for Weka. |
| trainingSet.csv | Contains, when going through the loop, each topic’s tweets’ attributes. |
| tree.model | The model created by TM2, based off a Weka classifier and the training set. |
| tweetSet.arff | The converted tweetSet.csv file for Weka. |
| tweetSet.csv | Holds the training set, the top 200 of each topic, of tweets and their attributes. Also has whether the tweet is relevant or not. |

\*Note: I accidentally got trainingSet & tweetSet confused in the code. tweetSet is the training set, while trainingSet is the topic’s tweets, re-made for each topic.

### Main

This package contains the main file, STIRS, and the Results, as well as the interfaces for Query Expansion and Relevance modules.

Java Files:

|  |  |
| --- | --- |
| Name | Description |
| Module | An interface method used among all “relevance” modules (e.g. Link Crawling, Machine Learning, etc). Makes sure that they all return the same kind of formats. All “relevance” modules should return all tweets used in the run, with those whom had their scores adjusted. |
| QueryExpansion | An interface method used among all query expansion modules (e.g. Google, Links, CommonWords, etc). Makes sure they all return the same format; a new arraylist, the size of the amount of total topics, which each part of the arraylist is a string of the new queries for that topic. Should be in right order to correspond to topics; i.e. MB001 first, MB002 second, etc. |
| Results | Creates the ouput.csv files, either in the format for STIRs to use for a run or in the submission format. |
| Stirs | The “main” class which all modules are run from. |

### Miscellaneous

This package contains items that are used across various places in STIRS.

Java Files:

|  |  |
| --- | --- |
| Name | Description |
| Normalizer | * After each “relevance” module is run, updates the ranks depending on the score. * Fixes up all the scores after all modules are run, so that they are all based off the same threshold. |
| POSTagger | Finds the parts of speech for a word. |
| TweetIDComparator | A class of type Comparator that allows one to sort by the ID of a tweet.  Example Code (for all comparators):  RankedTweetList currentList = tweets.get(i);  List<Tweet> sublist = currentList.getRankedList().subList(0, currentList.size());  Collections.sort(sublist, Collections.reverseOrder(new TweetScoreComparator())); |
| TweetRankComparator | A class of type Comparator that allows one to sort by the rank of a tweet. |
| TweetScoreComparator | A class of type Comparator that allows one to sort by the score of a tweet. |

### nistEvaluation

Used, after a run has been completed, to find how precise the run was for determining relevance. Finds Precison@30 (the top 30 tweets, for each topic, are searched through a NISTJudgment file).

Java Files:

|  |  |
| --- | --- |
| Name | Description |
| AllNistTopics | A “data structure” class; used to see how much of, for each topic, we considered relevant vs. what NIST actually said was relevant. |
| NISTTopic | A “data structure” class; returns an ArrayList. For each topic, it contains the non-relevant, relevant, and highly relevant tweets. |
| PrecisionScore | Returns a score, for each run, of how well we did compared to NIST (comparing, per topic, how many tweets were scored correctly) |

### Google

A package for External Query Expansion module Google (#3a), which Googles each topic, grabs the link content and finds common words for each topic.

Note: This is one of the “older” packages created original in 2011.

Java Files:

|  |  |
| --- | --- |
| Name | Description |
| Article | Removes all words that are not considered to be a common word:   1. Stop words 2. Query words 3. Urls |
| Downloader | The “main” class for Google. Finds the common words though the Content folder for each topic. If –updateContent is on, googles each topic and creates a content folder for it.  Note: If you’re updating the content, make sure to update the setGoogleLinks() variables for year, month1 and day1. These should correspond to the first date that the corpus started scrapping (for 2013, that was February 1, 2013). |
| GoogleExpansion | Goes through each url’s content for each topic and finds the top 4 common words among all urls for each topic. |
| Link | A “data structure” class used for holding Links downloaded by Google. |
| StopWords | Contains various list of removal words, self-made in 2011. Only used in Google module. |

Files:

|  |  |
| --- | --- |
| Name | Description |
| Content | Holds an index of content for each topic. Should only need to be updated once for every topic. |

### Wordnet

Used for External Query Expansion Wordnet (#3b), containing all the files used in the module.

Note: This is one of the “older” packages created original in 2011.

Java Files:

|  |  |
| --- | --- |
| Name | Description |
| LucQeWord | Finds synonyms for each topic given to it. |
| Phrase | Used in LucQeWord? Didn’t really get into this class, but it seems needed for Wordnet analysis. |
| PoST | Used in LucQuWord to find the parts of speech of a word, which helps when finding the best synonym that matches it. |
| QueryExpansionExternalb | The “main” class for running WordNet. |
| Word | Used in LucQeWord? Didn’t really get into this class, but it seems needed for Wordnet analysis. |

Files:

|  |  |
| --- | --- |
| Name | Description |
| Wordnet | Contains the download of Wordnet, which Wordnet needs to run Wordnet in the class. This can be found here: <http://wordnet.princeton.edu/wordnet/download/current-version/>. |

### Query\_ExpansionInternal

Used for Internal Query Expansion modules CommonWords (#3c) and HashTags (#3e).

Java Files:

|  |  |
| --- | --- |
| Name | Description |
| changeSlang | Finds all slang for all tweets in the run and replaces them with the equivalent real word. Uses slangDict.txt and creates output-slang.csv. |
| CommonWords | Used to find the top 10 words for each topic by going through the initial tweets found by the Lucene API. |
| expandedQuery | Uses a POSTagger, if wanted, to trim down on words found. |
| HashTags | Used to find the top hashtags for each topic. |
| QueryExpansionInternal | The “main” class for both CommonWords (#3c) and Hashtags (#3e). |

Files:

|  |  |
| --- | --- |
| Name | Description |
| englishStop.txt | Contains a list of stop words for removal. Created by Dr. Lim. |
| hashtags.txt | Created by Hashtags; contains all hashtags found for each topic. |
| slangDict.txt | Contains a list of slang 🡪 regular format. Created in 2011. |

### Relevance

Contains all “Relevance” modules; those who adjusts scores on individual factors, just as the number of retweets a tweet has.

Java Files:

|  |  |
| --- | --- |
| Name | Description |
| Google | Grabs top 10 tweets of each topic. Googles these tweets. If those links match up with the corresponding tweet topic, increases score.  Note: Think about, for future, if it would be better to find top 40 (due to precision@30); otherwise this just reinforces the top 10. But be careful – Google will kick you out if the program Googles too much too quickly. |
| GoogleInfo | A “data structure” class to hold the topic’s information. |
| rescoreTweets | Goes through the run’s tweets and adjusts the score based on 1) the number of retweets a tweet has and 2) the percentage of the tweet that contains the topic words. |

Files:

|  |  |
| --- | --- |
| Name | Description |
| Links.txt | Contains the tweet links, created by Google.java. |

### TTG

Contains the file(s) for the TTG task: "I have information need expressed by a query Q at time t and I would like a summary that captures relevant information." In this year's task, the summary is operationalized by a list of non-redundant, chronologically ordered tweets that occur before time t.

Java Files:

|  |  |
| --- | --- |
| Name | Description |
| TM3 | The “main” class for TTG. Goes through top 100 tweets, for each topic, of the run the scores the best precision. Then goes through each cluster, for each topic, and compares the percentage of the first tweet\* in the cluster to the current tweet. If the percentage is above the threshold given, adds to the cluster; otherwise creates its own cluster. Each first tweet of each cluster, for each topic, makes up the results returned for that topic.  \*First tweet makes up the “rules” for that cluster, so we only need to check that tweet  Note: We only go through the first tweet (which is the highest scored) for each cluster to print out, but perhaps there should be something that finds the best representation of that cluster? Future idea. |

## Twitter-Tools-Core

This is code provided by TREC for accessing the API. For more information, and to get the current code, check out: <https://github.com/lintool/twitter-tools/wiki/TREC-2013-API-Specifications>.

# Useful Links

Note: These links may be outdated.

TREC Microblog Track Related Links:

* <https://groups.google.com/forum/#!forum/trec-microblog>
* <https://github.com/lintool/twitter-tools/wiki/TREC-2014-Track-Guidelines>
* <http://trec.nist.gov/data/microblog2013.html>
* <http://trec.nist.gov/act_part/tracks.new11.html> (username and password needed; ask Dr. Small/Dr. Lim)
* <https://sites.google.com/site/microblogtrack/>
* <https://github.com/lintool/twitter-tools>
* <https://groups.google.com/forum/#!forum/trec-twitter-tools>
* <https://github.com/lintool/twitter-tools/wiki/Baseline-runs-using-the-TREC-2013-API>
* <https://github.com/lintool/twitter-tools/wiki/TREC-2014-Track-Guidelines>
* <https://github.com/lintool/twitter-tools/wiki/TREC-2013-API-Specifications>
* <http://www.umiacs.umd.edu/~jimmylin/trec2014microblog/servers.txt> (username and password needed; ask Dr. Small/Dr. Lim)
* <http://www.umiacs.umd.edu/~jimmylin/publications/>
* <http://trec.nist.gov/pubs/trec22/appendices/microblog.html>
* <http://trec.nist.gov/pubs/trec22/trec2013.html>
* <http://trec.nist.gov/proceedings/proceedings.html>

Eclipse-Related:

* <http://help.eclipse.org/indigo/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2Ftasks%2Ftasks-33.htm>
* <http://stackoverflow.com/questions/13125551/lucene-4-0-overrides-final-method-tokenstream>
* <http://archive.apache.org/dist/lucene/java/>
* <http://lucene.apache.org/core/4_7_0/core/org/apache/lucene/analysis/package-summary.html>

Wordnet:

* <http://lyle.smu.edu/~tspell/jaws/>
* <http://wordnet.princeton.edu/wordnet/download/current-version/>

Weka:

* <http://weka.wikispaces.com/Use+WEKA+in+your+Java+code>
* <http://stackoverflow.com/questions/20017957/how-to-reuse-saved-classifier-created-from-explorerin-weka-in-eclipse-java>
* <http://weka.wikispaces.com/Serialization>
* <http://stackoverflow.com/questions/10394615/unary-class-text-classification-in-weka>
* <http://stackoverflow.com/questions/13556062/weka-class-invalid-for-deserialization>

Other:

* <http://searchnewscentral.com/20110928186/General-SEO/how-to-scrape-search-engines-without-pissing-them-off.html>
* <http://google-scraper.squabbel.com/>
* <https://code.google.com/p/language-detection/>
* <http://stackoverflow.com/questions/4658606/import-existing-source-code-to-github>