SciTweets Documentation

By Marc D'Mello

Website url: scitweets.com (Note: This version of the website using the neural net it is only on my local machine, but this version has the same user interface and will show how SciTweets works)

GitHub: https://github.com/mdmarshmallow/SciTweets

What is SciTweets?

SciTweets is a website that goes through the Twitter timelines of scientists and scientific organizations and finds any links tweeted out. It then runs the articles (that the links lead to) through a neural net and returns a list of summaries for those articles.

Technology Used

Frontend:

- CSS
- HTML
- JSP (Java Servlet Pages)
- JSTL (Java Standard Tag Library)
- Javascript
- Bootstrap Classes

Backend:

- Java EE
- Maven

Web Server/Database:

- Webserver: Tomcat 8
- Database: MySQL 5

APIs/Libraries for Java

- Twitter4j: Wrapper for the Twitter API
- Aylien API SDK: Wrapper for the Aylien API which was used for summarization
- Jsoup: Library used to connect to websites and retrieve the HTML
- MySQL Connector: Allows the app to connect to the database

Neural Net Classifier

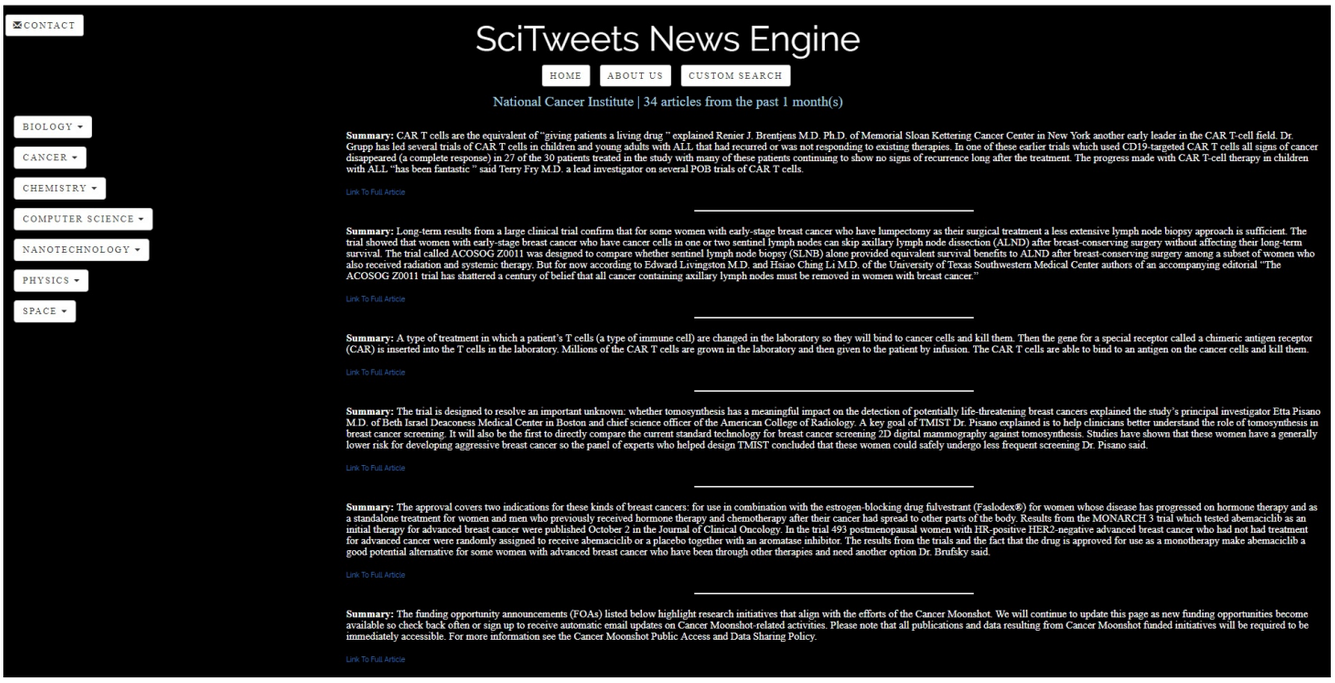
- Python: Used for TensorFlow API
- TensorFlow: Machine learning library I used to train, evaluate, and save the machine learning model
- Pandas: Library used to get data from a CSV file

News Engine / Custom Search

The two main features of SciTweets are the news engine and the custom search function. There is also an Administrator control section that will be covered in a later section.

News Engine:

The news engine looks through tweets of prominent scientists and organizations that are pre-set by an administrator. The following screenshot is an example of the what happens when National Cancer Institute is clicked:



An important thing to note for the news engine is that some handles load much faster than others. That is because if a handle is clicked on for the first time, all the links associated with that handle need to run through a filter. However, for all subsequent times the handle is clicked, most of the links won't need to run through a filter because they are stored in the database.

The following is `NewsEngineServlet` class (GitHub link to class):

```
@SuppressWarnings("serial")
@WebServlet(urlPatterns = "/newsengine")
//serves the newsengine.jsp
public class NewsEngineServlet extends HttpServlet {

	private TweetService tweetService = new TweetService();
	private CategoryService categoryService = new CategoryService();

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		//sets sends the categories list and the method to the jsp
		request.setAttribute("categories", categoryService.retrieveCategories());
		request.setAttribute("method", "get");
		request.getRequestDispatcher("/WEB-INF/views/newsengine.jsp").forward(request, response);
	}

	//doPost is called when a handle is clicked on the page
	protected void doPost(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		try {
			//gets a list of the tweets associated with that user (they are filtered in the retrieveTweets function)
			List<STweet> tweetList = tweetService.retrieveTweets(request.getParameter("user"));
			//adds the tweets and the number of tweets
			request.setAttribute("tweets", tweetList);
			request.setAttribute("numOfArticles", (tweetList).size());
			//gets the month and year of the oldest tweet on the list
			int month = tweetList.get(tweetList.size() - 1).getMonth();
			int year = tweetList.get(tweetList.size() - 1).getYear();
			//calculate months calculates the number of months from now and the first tweet
			int difference = calculateMonths(month, year);
			if (difference != 0){
				request.setAttribute("months", difference);
			} else {
				request.setAttribute("months", "");
			}
		} catch (SQLException | TextAPIException | TwitterException | InstantiationException | IllegalAccessException
				| ClassNotFoundException e) {
			//no error handling again :(
			e.printStackTrace();
		}
		//resends the categories and the sends the name of the tweet's author back
		request.setAttribute("name", request.getParameter("name"));
		request.setAttribute("categories", categoryService.retrieveCategories());
		request.setAttribute("method", "post");
		request.getRequestDispatcher("/WEB-INF/views/newsengine.jsp").forward(request, response);
	}

	//function calculates the difference between the current month and the inputted month given the year
	private int calculateMonths(int month, int year) {
		Date date = new Date();
		Calendar calendar = Calendar.getInstance();
		calendar.setTime(date);
		int currentMonth = calendar.get(Calendar.MONTH);
		int currentYear = calendar.get(Calendar.YEAR);
		//this is where the calculation happens
		int difference = (((currentYear - year) * 12) + currentMonth) - month;
		return difference;
	}
}
```

The `TweetService` object is important here as it retrieves the user's Tweets using the Twitter4j library. In the `NewsEngineServlet` class, that tweet retrieval occurs at this line of code:

```
List<STweet> tweetList= tweetService.retrieveTweets(request.getParameter("user"));
```

The following code shows the `TweetService` class (GitHub link to class):

```
//class that interfaces with the Twitter API
public class TweetService {
	private static List<STweet> tweets = new ArrayList<STweet>();
	private static TwitterFactory tf = new TwitterFactory();
	private static Twitter twitter = tf.getInstance();
	private String url;

	//retrieves the timeline of a user given the username
	public List<STweet> retrieveTweets(String username) throws IOException, SQLException, TextAPIException,
			TwitterException, InstantiationException, IllegalAccessException, ClassNotFoundException {
		if (!tweets.isEmpty()) {
			tweets.clear();
		}
		User user = twitter.showUser(username);
		long userID = user.getId();
		Paging p = new Paging();
		//only gets the past 100 tweets in the timeline
		p.setCount(100);
		List<Status> statuses = twitter.getUserTimeline(userID, p);
		List<String> urlsOnPage = new ArrayList<String>();
		List<String> summariesOnPage = new ArrayList<String>();
		//goes through each status
		for (Status status : statuses) {
			//checks for a url
			if (Filter.hasURL(status)) {
				//gets the url of the tweet
				url = status.getURLEntities()[0].getExpandedURL();
				/*checks if the url isn't already 'on the page' (not really on the page but put in the ArrayList that
				gets sent to the page)*/
				if (!urlsOnPage.contains(url)) {
					//deletes the second url of the tweet
					String statusText = ModifyTweet.deleteSecondURL(status.getText());
					//gets the date created and turns it into two ints to put in the STweet object
					Date date = status.getCreatedAt();
					Calendar calendar = Calendar.getInstance();
					calendar.setTime(date);
					int month = calendar.get(Calendar.MONTH);
					int year = calendar.get(Calendar.YEAR);
					int authorId = DBConnect.selectAuthorId(username);
					//checks if the url is already in the database
					if (DBConnect.selectFromLinkcache(url) != null) {
						//gets the summary stored in the database and the isValid boolean
						String description = DBConnect.selectFromLinkcache(url)[1];
						boolean isValid = DBConnect.checkIsValid(url);
						/*if isValid and the summary isn't already on the page, the tweets is
						added to the list of tweets to display*/
						if (isValid && !summariesOnPage.contains(description)) {
							tweets.add(new STweet(user.getName(), statusText, url,
											description, month, year));
							//these two are arrays that help avoid duplicate tweets/urls/summaries
							urlsOnPage.add(url);
							summariesOnPage.add(description);
						}
					//if it isn't in the database, runs the tweet through the filter
					} else if (Filter.checkTweet(url)) {
						//gets a 4 sentence long summary
						String description = SummarizeService.summarize(url, 4);
						/*if the summary isn't empty and the tweets that will be displayed don't already
						have the same summary*/
						if (description != null && !description.isEmpty() &&
						    !summariesOnPage.contains(description)) {
							//adds the information to the ArrayList
							tweets.add(new STweet(user.getName(), statusText, url, description, month, year));
							//stores the information in the database
							DBConnect.insertIntoLinkcache(url, description, authorId, true);
							//these two are arrays that help avoid duplicate tweets/urls/summaries
							urlsOnPage.add(url);
							summariesOnPage.add(description);
						} else {
							//puts puts the url in the database and marks inValid as false
							DBConnect.insertIntoLinkcache(url, null, authorId, false);
						}
					} else {
						//puts puts the url in the database and marks inValid as false
						DBConnect.insertIntoLinkcache(url, null, authorId, false);
					}
				}
			}
		}
		if (tweets.isEmpty()) {
			//if there were no valid tweets associated with the handle
			Date date = new Date();
			Calendar calendar = Calendar.getInstance();
			calendar.setTime(date);
			int month = calendar.get(Calendar.MONTH);
			int year = calendar.get(Calendar.YEAR);
			tweets.add(new STweet(null, null, null, "There seems to be no valid tweets :(", month, year));
		}
		return tweets;
	}
}
```

Custom Search:

The custom search function allows you to run any handle through the SciTweets filter. However, there are 2 catches to this. First, there is no summarization of the studies found. Second, none of the links are stored in the database so it will always take some time to load and list all of the tweets with scientific studies. The following is a screenshot of when '@naturephysics' is searched:

Here is the code for the custom search servlet (GitHub link to class):

```
@SuppressWarnings("serial")
@WebServlet(urlPatterns = "/customsearch")
public class CustomSearchServlet extends HttpServlet {

	private static TwitterFactory tf = new TwitterFactory();
	private static Twitter twitter = tf.getInstance();
	private String url;

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		if (request.getParameter("handlerequest") != null) {
			try {
				//stores the handle in the variable 'username'
				String username = request.getParameter("handlerequest");
				//gets the userId for later use
				User user = twitter.showUser(username);
				long userID = user.getId();
				//sets the tweets checked to the past 100 tweets
				Paging p = new Paging();
				p.setCount(100);
				//gets the user's timeline and stores it in an ArrayList
				List<Status> statuses = twitter.getUserTimeline(userID, p);
				List<STweet> tweetsToShow = new ArrayList<STweet>();
				//loop runs through the user's timeline
				for (Status status : statuses) {
					//checks for url
					if (Filter.hasURL(status)) {
						url = status.getURLEntities()[0].getExpandedURL();
						String statusText = ModifyTweet.deleteSecondURL(status.getText());
						//gets the date of the Tweet's creation
						//this isn't displayed on the JSP but it's still used in the STweet object
						Date date = status.getCreatedAt();
						Calendar calendar = Calendar.getInstance();
						calendar.setTime(date);
						int month = calendar.get(Calendar.MONTH);
						int year = calendar.get(Calendar.YEAR);
						//runs the url through the filter
						if (Filter.checkTweet(url)) {
							tweetsToShow.add(new STweet(user.getName(), statusText, url, null, month, year));
						}
					}
				}
				request.setAttribute("tweets", tweetsToShow);
				if (tweetsToShow.isEmpty()) {
					request.setAttribute("error", "noValidTweets");
				} else {
					request.setAttribute("error", "none");
				}
			} catch (TwitterException e) {
				//if the handles doesn't exist, sends and error in  the form of an attribute to the JSP
				if (((TwitterException) e).getStatusCode() == 404) {
					request.setAttribute("tweets", null);
					request.setAttribute("error", "handleNotFound");
				}
			} finally {
				request.getRequestDispatcher("/WEB-INF/views/customsearch.jsp").forward(request, response);
			}
		} else {
			request.setAttribute("error", null);
			request.setAttribute("tweets", null);
			request.getRequestDispatcher("/WEB-INF/views/customsearch.jsp").forward(request, response);
		}
	}
}
```

Filter / Neural Network Classifier

Filter Class

Here is the code for the filter class (GitHub link to class):

```
//this class has all the functions that identify if a Tweet contains a study
public class Filter {

	static RetrieveProperties rp = new RetrieveProperties();

	// checks to see if a tweet has a url using the Twitter4j URLEntity object
	// and the getURLEntities function
	public static boolean hasURL(Status status) {
		URLEntity[] url = status.getURLEntities();
		if (url.length != 0) {
			return true;
		} else {
			return false;
		}
	}

	// function retrieves the article and returns it as an ArrayList for easier
	// analysis
	private static String retrieveArticle(String urlInput) throws IOException {
		try {
			Document webpage = Jsoup.connect(urlInput).timeout(10 * 1000).ignoreContentType(true)
					.validateTLSCertificates(false).get();
			String article = webpage.body().text();
			return article;
			/*
			 * multiple catches for each error, not needed right now but will
			 * make it easier if error handling is implemented in the future for
			 * any of these errors
			 */
			// I found that this error is returned when there is a paywall on
			// the article
		} catch (HttpStatusException e) {
			return "paywall";
			// when Jsoup takes too long to connect to the url
		} catch (SocketTimeoutException e) {
			return "timeout";
			// I'm not really sure what causes a SocketException, hence the
			// generic 'badURL'
		} catch (SocketException e) {
			return "badURL";
			// Again, not really sure what happens with an SSL handshake
		} catch (SSLHandshakeException e) {
			return "SSLHandshakeException";
			// catch for all other exceptions
		} catch (Exception e) {
			return "Other Issue";
		}
	}

	// this function gets information from the article to put into the DNN classifier
	private static Map<String, Float> preProcessing(String article) throws FileNotFoundException {
		// splits the article into an array then gets the word count
		String[] wcArray = article.split("\\s+");
		float articleWC = wcArray.length;
		// loads the FilterWord.txt file to be used
		ClassLoader cs = new Filter().getClass().getClassLoader();
		File filterWordsFile = new File(cs.getResource("FilterWords.txt").getFile());
		Scanner scan = new Scanner(filterWordsFile);
		// checks how many words in the article are on the FilterWords file
		float wordCounter = 0;
		while (scan.hasNextLine()) {
			String word = scan.nextLine();
			if (article.toLowerCase().contains(word.toLowerCase()) && !word.isEmpty()) {
				wordCounter++;
			}
		}
		scan.close();
		// finds the average word length
		String characterCounter = article.replaceAll("\\s+", "");
		float numCharacters = characterCounter.length();
		float meanWordLength = numCharacters / articleWC;
		// finds what percent of the article is scientific
		float sciWordDensity = wordCounter / articleWC;
		// creates a mapping and returns it
		Map<String, Float> data = new HashMap<String, Float>();
		// SWC = scientific word count
		data.put("SWC", wordCounter);
		// WC = the article word count
		data.put("WC", articleWC);
		// AWL = the average word length
		data.put("AWL", meanWordLength);
		// SWD = the scientific word density
		data.put("SWD", sciWordDensity);
		return data;
	}

	// runs the the data from the map through a DNN Classifier (trained in TensorFlow)
	private static boolean checkArticle(Map<String, Float> data) throws IOException {
		try (SavedModelBundle bundle = SavedModelBundle.load(rp.getSciTweetsModelPath(), "serve")) {
			Session session = bundle.session();
			//creates tensors to input into the model
			Tensor<?> SWC = Tensor.create(new long[] { 1, 1 }, FloatBuffer.wrap(new float[] { data.get("SWC") }));
			Tensor<?> WC = Tensor.create(new long[] { 1, 1 }, FloatBuffer.wrap(new float[] { data.get("WC") }));
			Tensor<?> AWL = Tensor.create(new long[] { 1, 1 }, FloatBuffer.wrap(new float[] { data.get("AWL") }));
```

```
                        Tensor<?> SWD = Tensor.create(new long[] { 1, 1 }, FloatBuffer.wrap(new float[] { data.get("SWD") }));
                        //feeds each tensor in and retrieves the output
                        List<Tensor<?>> outputs = session.runner().feed("Placeholder_2:0", AWL).feed("Placeholder:0", SWC)
                                        .feed("Placeholder_3:0", SWD).feed("Placeholder_1:0", WC)
                                        .fetch("dnn/head/predictions/probabilities:0").run();
                        float[][] scores = new float[1][2];
                        outputs.get(0).copyTo(scores);
                        System.out.println(scores[0][1]);
                        //maps the output to true or false
                        int finalScore = Math.round(scores[0][1]);
                        return (finalScore == 1) ? true : false;
                }
        }

        // checks if the article has returned anything, and if so, runs it through the checkArticle function
        public static boolean checkTweet(String urlInput) throws IOException {
                String article = retrieveArticle(urlInput);
                if (article.length() != 0) {
                        if (!article.equals("paywall") && !article.equals("timeout") && !article.equals("badURL")
                                        && !article.equals("SSLHandshakeException") && !article.equals("Other Issue")) {
                                Map<String, Float> dataMapping = preProcessing(article);
                                return checkArticle(dataMapping);
                        } else {
                                return false;
                        }
                } else {
                        return false;
                }
        }
}
```

What this code does is interface with the saved machine learning model. In the `preProcessing()` function, SciTweets analyses the article and gets data about the article. This data includes the number of 'scientific words' (defined in a .txt file) in the article, the word count, the average word length, and percentage of words that are 'scientific words'. In the `checkArticle()` function, it creates tensors out of this data and inputs those tensors into the neural net classifier. It finally receives the output tensor and returns `true` or `false` based on that tensor.

Neural Network Classifier

The following is python code which I used to train, evaluate, and save the neural net classifier (GitHub link to code):

```
import tensorflow as tf
import pandas as pd

tf.logging.set_verbosity(tf.logging.INFO)


# loads the dataset into program
columns = ["SWC", "WC", "AWL", "SWD", "isValid"]
features = ["SWC", "WC", "AWL", "SWD"]
output = "isValid"
training_set = pd.read_csv(
    "scitweets_train.csv", skipinitialspace=True, skiprows=1, names=columns)
test_set = pd.read_csv(
    "scitweets_test.csv", skipinitialspace=True, skiprows=1, names=columns)


# define feature columns
feature_columns = []
for i in features:
    feature_columns.append(
        tf.feature_column.numeric_column(
            key=i))


# create classifier with hidden units 10, 20, 10 and saves it
classifier = tf.estimator.DNNClassifier(
    feature_columns=feature_columns,
    hidden_units=[10, 20, 10],
    model_dir="/scitweetsInfo/NeuralNetwork/scitweets_model")


# creates an input function that can be used with all the datasets
def get_input_fn(data_set, num_epochs, shuffle):
    """Generate and input function."""
    # creates a dict from the CSV file
    data = {}
    for j in features:
        data.update({j: data_set[j].values})
    # sets for input and output (x and y) in the input function
    x = pd.DataFrame(data)
    y = pd.Series(data_set[output].values)
    # creates input function
    input_fn = tf.estimator.inputs.pandas_input_fn(
        x=x, y=y, num_epochs=num_epochs, shuffle=shuffle)
    return input_fn


# trains the classifier in 10000 steps
classifier.train(input_fn=get_input_fn(
    training_set, num_epochs=None, shuffle=True), steps=10000)


# evaluates classifer and prints results
accuracy_score = classifier.evaluate(
    input_fn=get_input_fn(
        data_set=test_set, num_epochs=1, shuffle=False))["accuracy"]
print("\nTest Accuracy: {0:f}\n".format(accuracy_score))


# saves model so Java web application can access it
def serving_input_receiver_fn():
    """Build the serving inputs."""
    inputs = {}
```

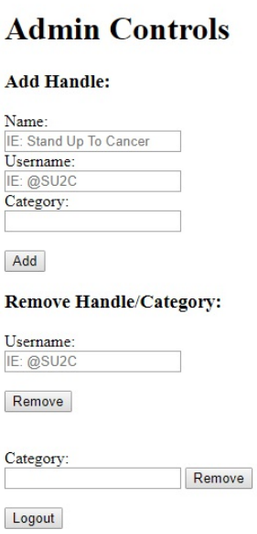
```
    for k in features:
        inputs.update({k: tf.placeholder(shape=[None, 1], dtype=tf.float32)})
    return tf.estimator.export.ServingInputReceiver(inputs, inputs)


classifier.export_savedmodel(
    export_dir_base="/scitweetsInfo/NeuralNetwork/scitweetsJavaModel",
    serving_input_receiver_fn=serving_input_receiver_fn)
```

This neural net classifier has 3 hidden layers with 10 neurons, 20 neurons, and 10 neurons respectively. It was written using a high level TensorFlow API called the tf.estimator API. It was then trained using a training set of data from 200 different links. I then evaluated it for accuracy with a set of data from 20 URLs (that were not part of the original training set). The neural net had a 90-95 percent accuracy rate with that training set. The training set for the neural net can be found at this link.

Administor Controls

The administrator controls allow handles and categories to be added and removed from the news engine. To get to the login page for the controls, you need to know the URL (scitweets.com/adminlogin) and enter the username and password. There is a filter preventing the access of the admin control servlets without a username in session. The admin controls page looks like this:



The admin controls allow the administrator to add and remove handles and categories, as you can see from the screenshot. This is a link to the GitHub `AdminControls` package.

Database

Overview

SciTweets uses a MySQL database. To create and setup the database, I used MySQL Workbench. A .sql file can be found on the SciTweets GitHub which sets up the database.

Uses

1. The database makes the website faster by allowing the website bypass the tweet filter process. When a link to a study is run through the filter, SciTweets stores the link into the database along with additional information like if the link is valid. With every url that is found in the initial tweet retrieval, a query is sent to the database for that url. If it is already stored in the database, no filtering needs to be done.
2. The database also stores summaries for every valid url, which helps to not hit the rate limits on the Aylien Summarizer API. When a url is found in the database, SciTweets will just use the already existing summary in the database rather than sending a request to the API.
3. Categories and handles are stored in the database

Tables

1. `linkcache` : This table stores URLs that are already filtered along with other information. The following screenshot shows the table with some information:

Id	url	summary	authorId	isValid	CreatedDateTime
1	http://www.mustardchallenge.com	NULL	1	0	2017-08-09 20:50:52
2	http://ow.ly/LiMb30e5mu9	Here is a snapshot of this promising resear…	1	1	2017-08-09 20:50:56
3	http://mustardchallenge.com	NULL	1	0	2017-08-09 20:50:56
4	http://bit.ly/2vNKb4D	NULL	1	0	2017-08-09 20:50:58
5	http://SU2C.org	NULL	1	0	2017-08-09 20:50:58
6	http://bit.ly/2wexwqX	NULL	1	0	2017-08-09 20:50:59
7	https://twitter.com/thedailybeast/status/89134…	NULL	1	0	2017-08-09 20:50:59
8	http://bit.ly/2vRfvi1	NULL	1	0	2017-08-09 20:51:02
9	http://cancerbikeride.org	Members of the BMS Oncology team will rid…	1	1	2017-08-09 20:51:05

2. `handles` : This table stores the name (which you give to a particular username) and the Twitter username of the handles you want to follow. It also stores some information associated with these handles. The following screenshot shows this table:

Id	name	username	CategoryID	CreatedDateTime
1	Stand Up To Cancer	@SU2C	1	2017-08-09 20:30:49
2	Nature Physics	@NaturePhysics	2	2017-08-09 20:30:49
3	Nature Biotechnology	@NatureBiotech	4	2017-08-09 20:30:49
4	Broad Institute	@broadinstitute	4	2017-08-09 20:30:49
5	NASA	@Nasa	7	2017-08-09 20:30:49
6	Node is	@Nodeis	5	2017-08-09 20:30:49
7	National Cancer Institute	@theNCI	1	2017-08-09 20:30:49

3. `category` : This table holds the names of the categories on SciTweets as well as the an associated number for each category. Here is how the table is set up:

CategoryID	CategoryName	CreatedDateTime
1	Cancer	2017-08-09 20:31:58
2	Physics	2017-08-09 20:31:58
3	Chemistry	2017-08-09 20:31:58
4	Biology	2017-08-09 20:31:58
5	Computer Science	2017-08-09 20:31:58
6	Nanotechnology	2017-08-09 20:31:58
7	Space	2017-08-09 20:31:58
NULL	NULL	NULL

Each table also has a trigger to fill the `CreatedDateTime` category.

Database Connection Class

The following code is part of the `DBConnect` class, which contains functions that interface with the database (GitHub link to class):

```
//this class contains all the functions that interact with the database
public class DBConnect {

	static RetrieveProperties rp = new RetrieveProperties();

	//connects to the database, used in all of the following functions
	private static Connection dbconnect()
			throws InstantiationException, IllegalAccessException, ClassNotFoundException, SQLException, IOException {
		Class.forName("com.mysql.jdbc.Driver").newInstance();
		Connection conn = DriverManager.getConnection(rp.getDBURL(), rp.getDBUsername(), rp.getDBPass());
		return conn;
	}

	//inserts the url, summary, tweet author, and an isValid bool which says if the link leads to a valid study
	public static void insertIntoLinkcache(String url, String summary, int authorID, boolean isValid)
			throws SQLException, IOException, InstantiationException, IllegalAccessException, ClassNotFoundException {
		Connection conn = dbconnect();
		PreparedStatement ps = conn.prepareStatement("INSERT INTO linkcache (url, summary, authorId, isValid) VALUES (?,?,?,?)");
		ps.setString(1, url);
		ps.setString(2, summary);
		ps.setInt(3, authorID);
		ps.setBoolean(4, isValid);
		ps.execute();
		conn.close();
	}
```

This class contains 12 methods, but only two functions are included here. The `dbconnect()` function establishes the connection with the database and is used in every other function in this class. The `insertIntoLinkcache()` function is similar to the other functions contained in this class.

Credits

Marc D'Mello: Wrote the backend (Java), set up the database, some HTML and javascript. Wrote and trained the neural net.

Adith Arun: Wrote the CSS, some HTML