# Twist:

# User Timeline Tweets Classifier


By

**Sonali Sharma**

**Priya Iyer**

**Vaidyanath Venkat**


## Code Documentation

Date: 12/06/2012.

**Mentor:**

Andy Schlaikjer

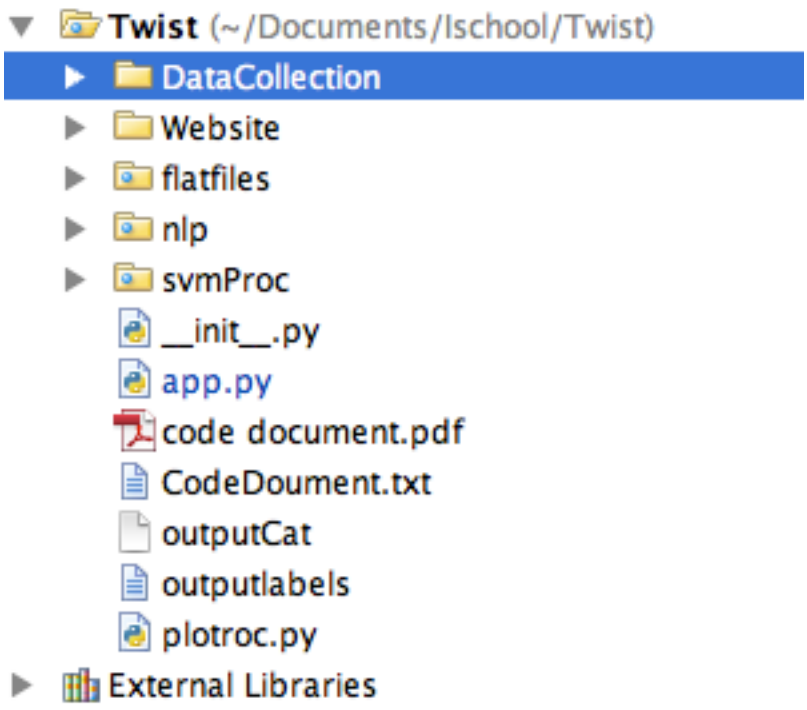
**Instructor**

Marti Hearst

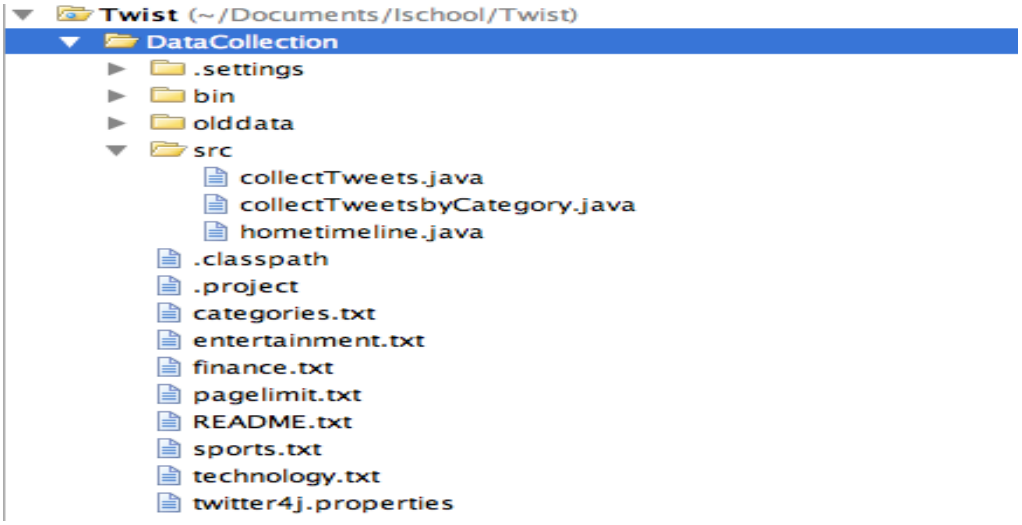
## Project Structure

Following is the overall structure of our code. It is divided into various modules based on the function.

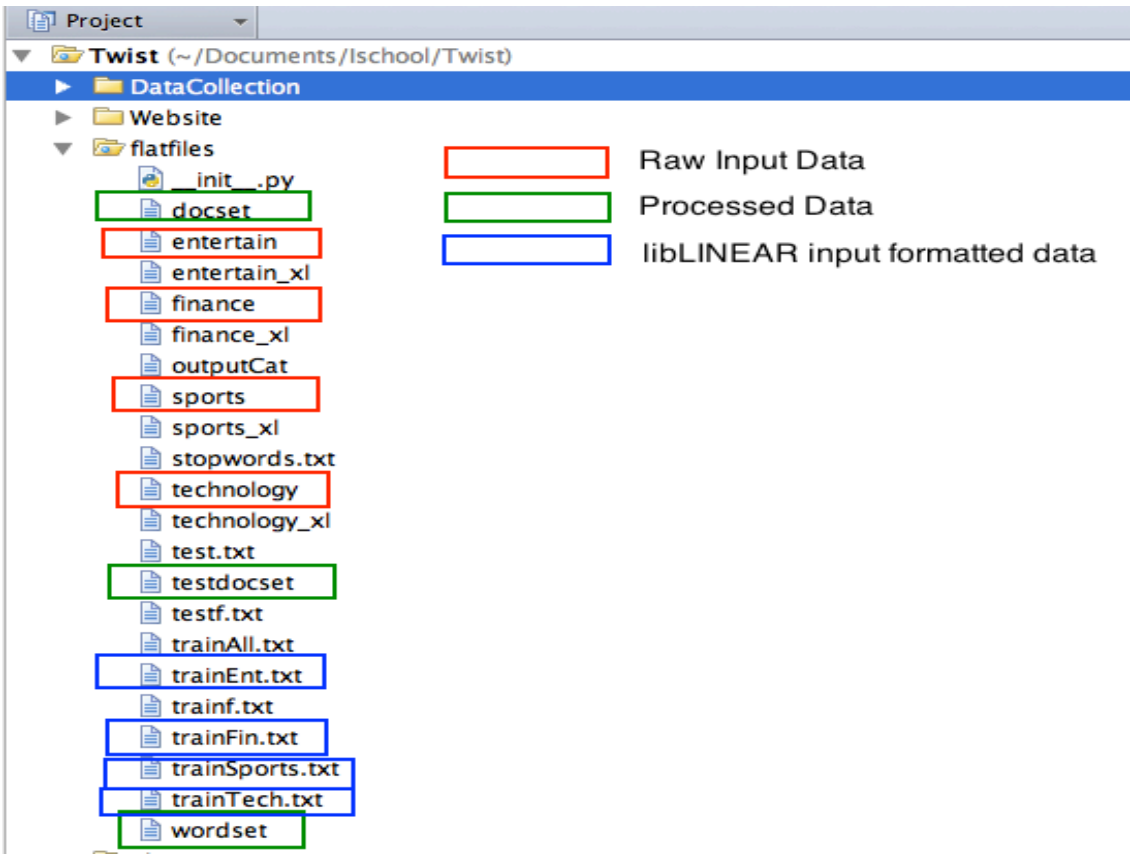
### Overall Structure



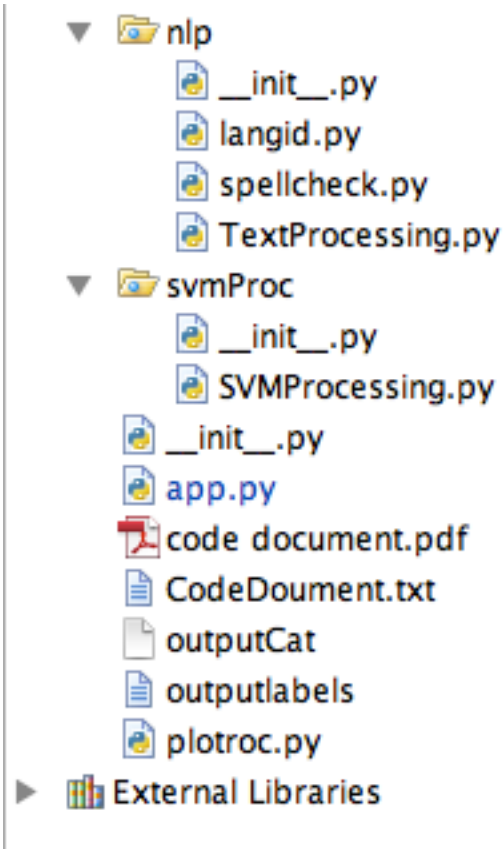
## Data Collection




## Flat Files



**NLP and SVM modules**




# Modules

This section provides detailed explanation of various modules used for our application

### Data Collection

We have used java with twitter4j package for data collection. We identified a list of influential users in each category from the who to follow list and fetched the tweets from those users.The twitter4j module allows us to access the REST API of twitter. Our module fetches tweets from user timeline fetching one page at a time(200 tweets) We used paging to evade the Rate limit on the REST api. The number of pages to be retrieved can

be configured using a configuration file(pagelimit.txt) which specifies the start and end page to be retrieved. By default this is set to 5 pages.
Please refer to the README.txt file in the DataCollection module.

### Website

This module contains the code and logic for the web interface that authenticate the user and classifies the tweets.
Flask is a python framework that is used to create web applications effectively using python. The basic structure of flask is to create an application file (python) that starts the server and details the routing logic and the corresponding html files to be rendered. These HTML files are saved in a seperate folder called as "Templates" and the necessary java script and style sheet files are in the "Static" folder.
For the authentication, we have used the FlaskOAuth library that has the basic features required to connect to twitter's authentication module and retrieve the authentication key.
Once the user is authenticated, and the user authorizes our app to access their tweets, this authentication code has to be saved in the database so that the same user is not required to authenticate every time. Instead of a database, we have opted to save the information in the app servers session memory and then reuse it throughout the session.
We specify the callback url for redirection after the authentication as localhost:5000/tweet. We can see the corresponding code for this module in the app.route('tweet') section of the code.
In Flask, the routing logic is specified by defining methods for each route and identifying them with the app.route() decorator. For example

```
app.route('/') # specifies the root path (http://localhost:5000)
 def get():
   #code for get
```

The functionalities in this module include the following

1. Authenticate the app and connect to twitter API , Use the Oauth module to create a twitter authentication object.
2. Redirect to the twtter Oauth page and get the authorization token for the user
3. Save the users credentials in session so that once authorized the app can use the users details repeatedly
4. Redirect to the callback url (app.route('tweet') )
5. Fetch user time line and select all tweets which are not posted by the authenticating user
6. Classify the tweets and show them on the interface

### App.py

This is the entry module, which is used to run the application. It enables us to either trains or classify. We used it for our own purpose of training and testing the classifier. We use option 1 to train the classifier and option 2 to test the classifier. These choices call one of the methods defined below:

| Method | Description |
|---|---|
| def classify(flag=None): | This methods calls a series of methods to test the classier. It starts with calling module for text processing , storing files data in data structure, create test file. |
| def train(): | This method calleD a series of other methods to do the following:<br>Text processing<br>Store file data in data structure<br>Create train file<br>Train libLINEAR |

## NLP

All the text processing code is contained in this module.

### TextProcessing.py

This is the interface that is called during the trainign and classification. From here the calls to the spell check module is made

1. Generate Unigrams
2. Filter punctuations.
3. Reduce strings. spell correct, stem
4. Generate bigrams
5. Stop words removal
6. Language Identification

| Method | Description |
|---|---|
| Process(flow) | This method processes a tweeet file and generates the word index and other necessary set of files required for training and for the actual classification. |
| removepunctuations(unigrams) | Takes in a list of words and removes all words that adoesnt satisfy the criteira which includes<br>1. No punctuations excpet for ' - and #<br>2. No urls. |

### Spell check.py

This contains the logic needed to implement the Norvig's spell check and other text cleaning.

1. Determine Part of speech and return all proper nouns without any check
2. Detemrine if any word contains the same letter more than three times consecutively (ex : cuuuuttteee)

3. Reduce such strings to their normal spelling.
4. For all words, compare with NLTK's dictionary and run spell check if needed.

Spell check:

1. For each word, split the word into 2 at all possible positions. - Ex -> hell is split as (h,ell), (he,ll), (hel,l), (hell,) etc
2. This becomes the input for the remaining steps
3. For each pair in the list we do the following
    a. Add a new letter from a though z at every possible position. So 'hell' yields ahell,bhell...zhell, haell,hbell..hzell etc.
    b. Edit each letter and substitute that with a through z. Ex, 'hell' yields aell,bell,...zell, hall,hbll..hzll etc..
    c.Delete each letter. so hell becomes, ell, hll, and hel etc.
    d. swap every pair of letters. For ex -> hell yields ehll, hlel, hell etc.
4. Create a set of distinct suggestions after these 4 operations.
5. Filter out all invalid words like 'aell'.
6. returns the final list.

| Method | Description |
|---|---|
| def spellcheck(unigram): | Takes a word as input and runs a spell check on the word |
| def findSuggestions(word): | This method contains the core logic for spell check. Finds suggestions and returns the list back |
| split(word) | Splits a word into parts (hell → (h,ell) etc. ) |
| deleteLetters(subStrings) | Deletes individual letters (hell → ell, hll etc) |
| swapLetters(subStrings) | Swaps two consecutive letters (hell → hlel, ehll etc) |
| editandInsertLetters(subStrings) | Edits each letter and replaces with alphaberts from a-z, inserts letters from a-z in all possible positions (hell → aell,bell,.. haell,hbell etc) |

**LangId.py**

This contains the logic as suggested by Misja Hoebe to use nltks trigram sets to detect language

1. Nltk has trigram sets for many languages with their wieghts in their corpus
2. From the corpus load the weights for english, french spanish
3. From the tweets, genrate trigrams and their frequencies.
4. This is then compared with the nltk corpus for a language match.

| Method | Description |
|---|---|
| LangDetect(object) | Entry point that works on the current word that invoked it and returns a predicted language |
| def detect(self, text): | Has the core logic to detect the language |
| def get_word_trigrams(self, match): | Get trigram matches from nltk corpus |
| def _get_trigram_weight(self, line): | Gets the weight of the trigram fom the corpus |
| def _read_trigram_block(self, stream): | fetches the nltk corpus and reads blocks of trigrams and gets the weight of each trigram |

## SVM

This directory contains the ML code required to classify the tweets.

**SVMProcessinsg.py**

Contains the core logic that trains and creates the model for classification, classifies the tweets and finally calculates metrics for the classification.

1. Loads the tweets into a global cache. This contains the tweet and word ids for all tweets in the set.
2. Creates different models by varying the cost and other parameters and trains a set of tweets.

3. Calculates the metrics for each model by computing the precision and recall values and decides on the apt model to be used.

4. Once the model is decided, the tweets can be classifed using the model.

5. Liblinear package is used to train the machine.

| | |
|---|---|
| def cacheTweetsInList(maxWords, maxTweets,flow): | This method reads the preprocessed training set data/test set data (based on whether we are training or testing) then stores it together in a data structure which is then used in the rest of the code. |
| def createTrainFile(docwords,docCatIds,maxTweets): | This creates an input file in the format of the liblinear input to train the classifier. |
| def createTestFile(docwords,docCatIds,maxTweets): | This creates an input file in the format of the liblinear input to test the classifier. |
| def trainlibLinear(): | This modules accepts a training data in the format discussed above and calls the following two methods of liblinear library:<br>1) train(labels,features,str(options)) – This method trains the classifier by taking the labels and features from the input file and of options for cost and type (as discuused in report). The output is a classifier.<br>2) predict(labels, features, m1) – This gives the methods was used to get the predicted values based on the classifier created using train and returns the predicted labels, features and accuracy measure. |
| def testSVM(flag=None): | This module accepts the test file in the liblinear format, passes it to the predict() method of liblinear library for each of the 4 classifiers and gets the predicted labels for the test set from al 4 models in sepaparte lists |
| def svmOutput(): | This reads the list output (which is a set of 4 labels for each tweet) from the previous step then determines the final category of the tweet. |