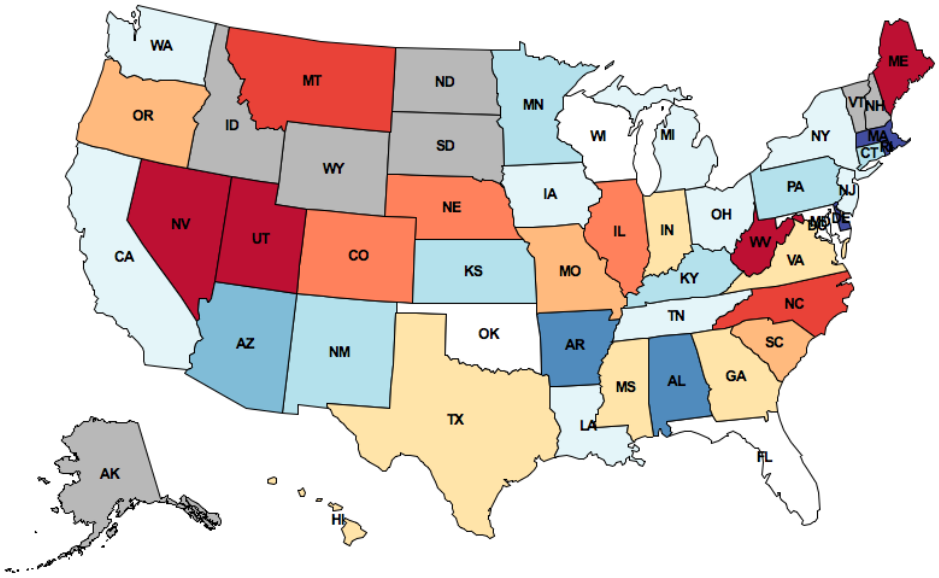
Ex7: Twitter Trends

Submission deadline - Monday, 16.12.2013, 20:55



What do people tweet?
Draw their feelings on a map
to discover *trends*.

Introduction

In this project, you will develop a geographic visualization of twitter data across the USA. You will need to use classes and different containers to create a modular program, and also implement few simple computational geometric algorithms.

The map displayed above depicts how the people in different states feel about Texas. This image is generated by:

1. Collecting public Twitter posts (tweets) that have been tagged with geographic locations and filtering for those that contain the "texas" query term.
2. Assigning a sentiment (positive or negative) to each tweet, based on all of the words it contains.
3. Aggregating tweets by the state with the closest geographic center, and finally
4. Coloring each state according to the aggregate sentiment of its tweets. Red means positive sentiment; blue means negative.

The details of how to conduct each of these steps is contained within the project description. By the end of this project, you will be able to map the sentiment of any word or phrase.

A small version that contains all the starter code, but only a small subset of the data can be downloaded from here. You can complete the project in its entirety using this archive. Use this link to download the full list of tweets (**warning: 81 MB**, unzip and place in the data directory) so you can play with more terms.

The project uses several files that **you are not allowed to change**:

- trends.py - The main project file.
- geo.py - Geographic positions, 2-D projection equations, and geographic distance functions.
- maps.py - Functions for drawing maps.
- data.py - Functions for loading Twitter data from files.
- graphics.py - A simple Python graphics library.
- ucb.py - Utility functions.
- Other files for testing

In addition, we provide you three files with starter code. **All your coding will be in the following files**:

- tweet.py – A class to represent Tweets.
- geo_tweet_tools.py - Functions for finding where the tweets where tweeted from.
- nation_mood.py - Functions for summing the mood of the states.

The data directory contains all the data files needed for the project.

Phase 1: The Feelings in Tweets

In this phase, you will create a class to represent tweets.

Tweets

Problem 1. In the file **tweet.py** complete the implementation for the class Tweet. An instance of the type Tweet is initialized with the following parameters:

- `text`: a string, the text of the tweet.
- `time`: a datetime object, when the tweet was posted.
- `latitude`: a floating-point number, the latitude of the tweet's location.
- `longitude`: a floating-point number, the longitude of the tweet's location.

Implement the following methods:

- `get_text()` - Returns the text of the tweet.
- `get_time()` – Returns a `datetime` object, when the tweet was posted.
- `get_location()` – returns a `Position`. Position is a class defined at the top of **geo.py**. Make sure that you understand how to manipulate positions; they play an important role in this project.
- `get_words()` – Returns an ordered list of all words in the tweet. We define a "word" as any consecutive substring of `text` that consists **ONLY** of ASCII letters.

The string `ascii_letters` in the `string` module contains all letters in the ASCII character set. Words should be converted to lowercase.

- `get_sentiment(word_sentiments)` – Returns the sentiment of the tweet, that is, the average sentiment of all the words in the tweet that have a sentiment. Returns **None** if none of the words have a sentiment.
  Use the dictionary `word_sentiments` to get a sentiment for a specific word - The `word_sentiments` dictionary maps words to values (between -1 and 1), which represent negative and positive feelings.

When you complete this phase, try running the following line:

```
python3 trends.py -p <sentence>
```

with different texts to get the sentiment of the sentence.

Phase 2: The Geometry of Maps

Positions

In this phase, you will write two functions that together determine the centers of U.S. states. The shape of a state is represented as a list of polygons. Some states (e.g. Hawaii) consist of multiple polygons, but most states (e.g. Colorado) consist of only one polygon (still represented as a length-one list).

We will use the `Position` class to represent geographic latitude-longitude positions on the Earth.

Problem 2. In the file **geo_tweet_tools.py** implement the function `find_centroid`, which takes a polygon and returns a tuple - the `Position` of its centroid and its area. The input polygon is represented as a list of `Position` instances, which are the consecutive vertices of its perimeter. The first vertex is always identical to the last.

The centroid of a two-dimensional shape is its center of balance, defined as the intersection of all straight lines that evenly divide the shape into equal-area halves. `find_centroid` returns the centroid and area of an individual polygon.

The formula for computing the centroid of a polygon appears on Wikipedia. The formula relies on vertices being consecutive (either clockwise or counterclockwise; both give the same answer), a property that you may assume always holds for the input.

The area of a polygon is never negative. Depending on how you compute the area, you may need to use the built-in `abs` function to return a non-negative number.

When you complete this problem, the doctest for `find_centroid` should pass:

```
python3 trends.py -t find_centroid
```

Problem 3. In the file **geo_tweet_tools.py** implement the function `find_center`, which takes a shape represented by a list of polygons and returns a `Position`, its centroid. You may assume that the area is not 0.

The centroid of a collection of polygons can be computed by geometric decomposition. The centroid of a shape is the weighted average of the centroids of its component polygons, weighted by their area.

When you complete this problem, the doctest for **`find_center`** should pass:

```
python3 trends.py -t find_center
```

Once you are finished, **`draw_centered_map`** will draw the 10 states closest to a given state (including that state):

```
python3 trends.py -d CA
```

States

The name **`us_states`** is bound to a dictionary containing the shape of each U.S. state, keyed by its two-letter postal code. You can use the keys of this dictionary to iterate over all the U.S. states.

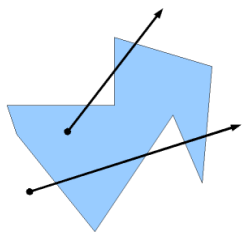
In this phase, you will write functions to determine the state that a tweet is coming from, group tweets by state, and calculate the average positive or negative feeling in all the tweets associated with a state.

Problem 4. In the file **geo_tweet_tools.py** implement the function **`find_closest_state`**. This function takes a dictionary of the states centers, and returns a reference to a function, let's call it **`find_state`**. **`find_state`** is a function that takes a Tweet and returns the two-letter postal code of the state that its center is the closest to the location of the tweet (Hint: use nesting). Use the **`geo_distance`** function (provided in **geo.py**) to calculate the shortest distance in miles between two positions.

When you complete this problem, the doctests for **`find_closest_state`** should pass:

```
python3 trends.py -t find_closest_state
```

Problem 4 ½ (BONUS - 5 points). In the file **geo_tweet_tools.py** implement the function **`find_containing_state`**. Like **`find_closest_state`**, this function returns a function that takes a Tweet and returns the two-letter postal code of the state that **contains** the location of the tweet.



The problem of deciding if a point is inside or outside a polygon is known as the "Point in Polygon" (PIP) problem, and there are two common approaches to solve it. The simplest is called the "Ray-casting algorithm" and it based on a simple observation:

1. Find a ray that starts from the point in question and goes in ANY fixed direction that intersects the polygon borders.
2. If the number of intersections is an even number than the point is outside the polygon, and if it is odd it is inside.

See the above links for further details and implementations of the algorithm.

When you complete this problem, the doctests for **`find_containing_state`** should pass:

```
python3 trends.py -t find_containing_state
```

Problem 5. In the file **geo_tweet_tools.py** implement the function **`group_tweets_by_state`**, which takes a sequence of tweets and a reference to function, and returns a dictionary.

The referenced function (**`find_state`**) is a function that receives a tweet and returns a state name (two-letter postal codes). The keys of the returned dictionary are state names (two-letter postal codes), and the values are lists of tweets that are mapped to the state using the **`find_state`** function.

When you complete this problem, the doctests for **`group_tweets_by_state`** should pass:

```
python3 trends.py -t group_tweets_by_state
```

Phase 3: The Mood of the Nation

Problem 6. As an exercise, in the file **nation_mood.py** implement the function **`most_talkative_state`**, which returns the state containing the most tweets containing a given term. If no tweets return None.

When you complete this problem, the doctests for most_talkative_state should pass:

```
python3 trends.py -t most_talkative_state
```

Problem 7. In the file **nation_mood.py** implement the function **`average_sentiments`**. This function takes the dictionary returned by **`group_tweets_by_state`** and also returns a dictionary. The keys of the returned dictionary are the state names (two-letter postal codes), and the values are average sentiment values for all the tweets in that state.

If a state has no tweets with sentiment values, leave it out of the returned dictionary entirely. Do not include a state with no sentiment using a zero sentiment value. Zero represents neutral sentiment, not unknown sentiment. States with unknown sentiment will appear gray, while states with neutral sentiment will appear white.

You should now be able to draw maps that are colored by sentiment corresponding to tweets that contain a given term:

`python3 trends.py -m texas` (the colors should be as the picture at the top of the file)

```
python3 trends.py -m israel
python3 trends.py -m basketball
python3 trends.py -m computer
python3 trends.py -m sandwich
```

If you downloaded the small version of the project, you will only be able to map these five terms. If you would like to map any term, you will need to download this twitters data file and place it in the **data** directory of your project.

In this last section, you will take into account the fourth dimension: time. Each tweet has a **`datetime`** object, which represents the time that the tweet was posted.

The provided **`draw_map_by_hour`** function visualizes the tweets that were posted during each hour of the day. For example, "sandwich" tweets appear most positive at 10:00pm: late night snack!

Problem 8. In the file **nation_mood.py** implement the function **`group_tweets_by_hour`** which takes a sequence of tweets and returns a list of lists. The indexes of the returned list represent the 24 hours of the day. The inner lists are lists of the tweets that were posted during that hour.

To get started, read the online [documentation] on **`datetime`** objects. When you are finished, you can visualize the changing sandwich sentiments over time.

```
python3 trends.py -b sandwich
```

Running the bonus: If you implemented the bonus you can add the argument 'c' so the program will use **`find_containing_state`** instead of **`find_closest_state`** to determine a tweets location. For example:

```
python3 trends.py -cm israel

python3 trends.py -cb israel
```

Acknowledgements: This project is based on an exercise developed by Aditi Muralidharan John DeNero and Hamilton Nguyen.

Submission and Further Guidelines

<u>README</u>

Explain in the usage part how to run the project and about the different options.

<u>Creating a tar file</u>

You should have edited the files:

- **tweet.py**
- **geo_tweet_tools.py**
- **nation_mood.py**
- **README** (as explained in the course guidelines)

Create a TAR file named ex7.tar containing only the above files by invoking the shell command:

```
tar cvf ex7.tar tweet.py tweets.py nation_mood.py README
```

The TAR file should contain only these files!

<u>Submitting the tar file</u>

- You should submit the file ex7.tar via the link on the course home page, under the ex7 link.
- Few minutes after you upload the file you should receive an email to your university mailbox. That email contains the test results of your submission and the PDF file that was passed to the grader.
- Read the submission file again and make sure that your files appear and fully readable.

<u>Running the tests yourself</u>

There are two ways you can run these testers and see the results without submitting your exercise and getting the PDF:

The first – place your tar file in an empty directory, go to that directory using the shell and type: ~intro2cs/bin/testers/ex7 ex7.tar

The second – download a file – ex7testing.tar.bz2. Extract the contents (using tar –xjv ex7testing.tar.bz2) of the file and follow the instructions on the file named TESTING.

<u>School Solution</u>

Can be found in *~intro2cs/bin/ex7/trends*. You can see all the possible options by invoking the h option. <u>Unless using the T option, it will put the data files by words in the working directory</u>.

***Congratulations!* One more Intro2cs project completed.**