* The Whale

Visualizing Twitter with local groups
Erik Turnquist, Rob Hanlon, Eric Kimbrel, Zack Allred
Overview

All features we could possibly think of:
- Visualization of Twitter relationships
- Friend and follower similarity
- Determine trends among a group of users
- Follower recommendations
- Identify groups of similar users

Core functionality achieved:
- visualization
- similarity
- trends
Architecture

Ruby
- jruby for requesting data
- OAuth for login and authorization
- `<canvas>` for UI

Java
- Auto-update twitter data (Erik)
- twitter4j
- Similarity ranking (Zack)
- Group trends (Eric)

MySQL
- Stores users, twitter users, tweets, trends, wordcounts
Gathering Data

- Background service for automatically updating data
  - Lists of friends and followers
  - Newest tweets
- Optimized for performance
  - Threadpools for API calls
  - Only receives new information from API
- Uses minimal API calls
- Twitter API Issues
Similarity

- Treat each user as a concatenation of their tweets.
- We tried two different algorithms to rate the similarity scores of the result.

- First Attempt:
  - Euclidean distance of the tf vectors (as proportions).
    - Ignored words whose frequency were too large or small
    - Ignored words used by only one of the two Twitterers
  - Similarity = 1 - the Euclidean distance

- Second Attempt:
  - Cosine similarity of the tf-idf vectors

We chose to use the cosine similarity of the tf-idf vectors since the first method did not take into account the document frequency of the terms.
Finding Trends on Selected Users.

1. Word Usage: Base trends off the most common words that occur between all selected users.

   Group of 3 users:
   hcr, house, healthcare, politics, base64, facebook, red, twitter, glenn, songoftheday

   Group of 254:
   db09, splatter, fb, hc09, sandvoxtip, unix@40, gomachine, dinner, unixtour, l4d

2. Term Extraction: Base trends on topics pulled from text using Yahoo's term extractor

   Group of 3:
   iphone, google, bandstand, sounders, storytelling, curio, dingell, yesss, bystander, skerik

   Group of 254:
   iphone, google, twitter, rt, tweets, tweet, tinyurl, ly, facebook, followers
Phase 1: Consider all of a user's tweets as a single document and translate the document into word counts or a list of topics.

Phase 2: When trends are requested, query our database and return the most frequent terms from the selected users.
Frontend structure

Non-GUI components:
- JRuby on Rails
  - Why? Direct construction and reference to Java objects from Ruby code, so Rails can do what it does best, and Java can do the number crunching
- twitter-auth RubyGem
  - 2 minute Twitter OAuth integration
- Resque
  - Used to asynchronously hit the backend and check for updated data
- Distributed Ruby
  - Conveniently shuttles around object references from Rails to Resque in order to make sure we're all working with the same objects

GUI components:
- `<canvas>`
  - HTML5
  - Open technology, new standard
  - Flash sucks, but canvas isn't really ready for primetime. Learned that the hard way
- Processing.js
  - Ported from Java by John Resig (of jQuery fame), a really intuitive framework for building animations and graphics on top of canvas
- traer.physics.js
  - Also ported from Java, but by me. :) Great for building force-directed layouts
Some eye candy