Improving Named Entity Recognition accuracy on Social Media Data

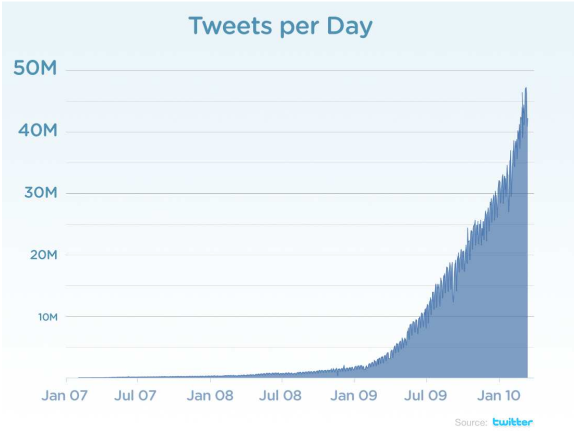
Will Murnane
willm1@cs.umbc.edu

May 19, 2010

The Domain: Social media data

- Short, informal "status messages"
- Many topics: day-to-day events, politics, sports, events, music, movies, almost anything
- Twitter, Facebook, Google Buzz, et al.
- Noisy: bad spelling, capitalization, punctuation
- Frequent: Easy to write, so many updates may be posted per day.
- Close to the action: people can send these messages from portable devices, so they often beat traditional news media to a story. (UK elections, for example.)

Pretty Big


Tweets per Day
50M
40M
30M
20M
10M
Jan 07
Jul 07
Jan 08
Jul 08
Jan 09
Jul 09
Jan 10
Source: twitter

The Problem: Information Extraction from Twitter

- There's a lot of useful data on Twitter, but it's hard for computers to get at.
- Traditional genres of text have different structure from Twitter, and this throws off a lot of machine learning algorithms.
- We want to tackle these problems in this new domain, too, and most of the problem is in re-training the existing algorithms.
- Features are weighted differently on Twitter, but a paper that attempted to cross-train across domains didn't work, so we'll train from scratch.

The Problem: Named Entity Recognition

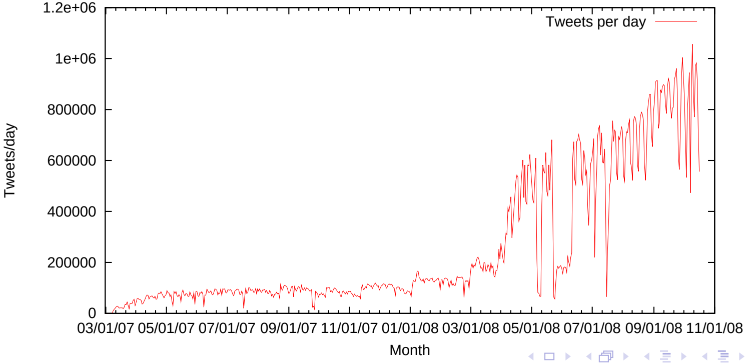
- Which words refer to entities: people, places, organizations, products, events?
- Existing techniques use machine learning to find these words
- This requires training data
- Training data can be generated by experts, but that's expensive and slow: dozens of hours of work, and experts aren't cheap.
- Also of interest, but not addressed here: which entities do these words refer to?

My contribution

- Implement a language detection algorithm and test its effectiveness on Twitter data
- Develop and evaluate an algorithm for ranking workers that includes both their agreement with other workers and their agreement with an expert-generated corpus.
- Compare a machine learning algorithm trained on a set of annotations selected using the algorithm to a corpus from another domain.
- Build a better NER system

Dataset Overview

- Contains about 150 million status updates from Twitter
- Collected over 20 months
- Rate of collection much higher in later months:



Dataset Organization

- Originally started as a MySQL dump of a production database
  - No foreign-key constraints enforced
  - Weird character encoding
  - Some characters encoded with HTML entities
- Converted to PostgreSQL and fixed all those problems
- Good for future work: no more cleaning up required
- Also built a Lucene index: fast keyword searches. This was already used to do some sentiment analysis.

Twitter conventions

Twitter statuses have some features that make them interesting (and in some ways, easier) to analyze.

A hashtag used as a word:

@pydanny	@kantrn I think need to have a cartwheel/au open space at next year's *#pycon*

A hashtag used as an event tag:

@Jess_Clarke	Another earthquake? Maybe that guy was right! *#boobquake*

A username mention:

@lovely_bieber23	I never knew *@BarackObama* had twitter..

An example retweet:

@chrisvargas1111	*RT @ConanObrien:* Who would invest in Goldman Sachs mortgage investments? I played it safe and bought Greek bonds and magic beans.

A URL, shortened by a third-party service:

@InsideAxis	Tech Breakthroughs: Springtime For Nukes: Previous springs have brought bad news to nuclear power advocates. Not s... *http://bit.ly/avMTcy*

These features can help pick out people (by looking for @-tags), places and events (looking at hashtags), and get more context about a tweet by looking at the text a URL points to.

Language detection algorithm

- Developed by Cavnar & Trenkle (1994)
- Used in industry: original implementation by Maciej Cegłowski in Perl, TextCat, KDE implementation in C++, later Python version
- Re-implemented in Java to fit in with the rest of the tools used
- 99+% precision on sample of 1000 tweets, recall less than dazzling (e.g., many English tweets marked as being in Scots)
- Iterative training on a Twitter-specific training corpus would be cheap and might help

- Essentially n-gram based: count all the 1-, 2-, 3-, and 4-grams in a sample of each language, and remember the most common few hundred (these lists can be stored on disk, only order matters).
- Now, to profile a sample of text in an unknown language, use the same technique to generate its list of n-grams.
- Then compare the sample's list to each of the known lists, by counting how far out of place each n-gram in the sample is in the known list. The sample is guessed to be in the language which minimizes the total distance.
- Easy to train: just need samples of text in several input languages. Current model based on short pages from the Tenth International Unicode Conference website, and it's still pretty good.

Overview of CRF

- The original CRF paper is from 2001, and the Stanford NER system is from 2005
- CRF is state-of-the art: fully generalized model.
- Stanford uses a reduced model called linear-chain CRF with transition values rather than probability distributions, which are easier to update and makes calculating the most likely state sequence less expensive (used when tagging a sequence).
- What about out-of-box models? Stanford includes two, but they do a poor job on Twitter data.

Other NER systems considered

- Mallet: more recent CRF system, runs fast, but doesn't include feature extractors.
- LingPipe: uses an older technique called Hidden Markov Models.
- NLTK: Big and complete... but no CRF, and not easy to get started with.
- Illinois Named Entity Tagger: no CRF, and it uses gazetteers which would be unsuitable for Twitter data.

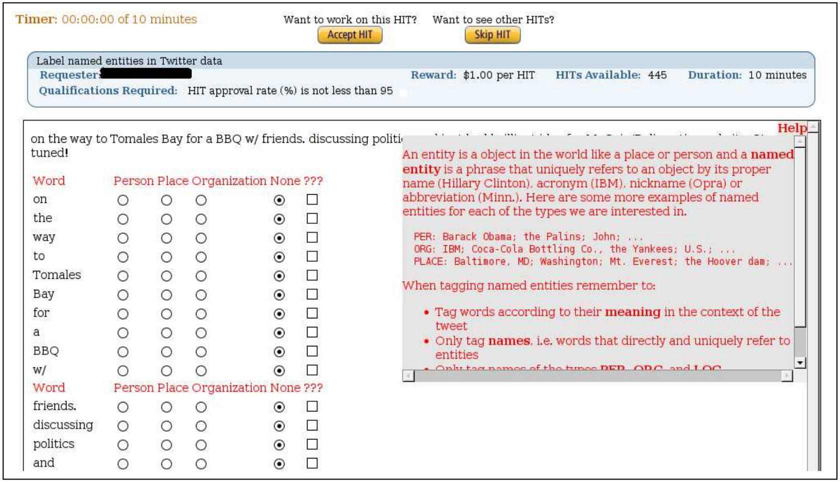
The Solution? Mechanical Turk

- Amazon's "artificial artificial intelligence": easy way to get humans to do repetitive tasks that require human judgement.
- Low startup overhead: need to know some HTML, and get your data into CSV files
- Fast: many workers can work at once on your task
- Cheap: workers earn a per-job fee which can be on the order of a few dollars per hour.
- Bad: workers are paid per job that they complete satisfactorily, so there's an incentive to complete jobs as quickly as possible rather than do a careful job, and even when they're careful they're not experts on your task.

The Solution? Mechanical Turk

- Repetitive tasks done by anyone who understands the questions, and some who don't.
- Bad accuracy for several reasons: rush to earn money, some bots auto-fill, some non-native speakers.
- Bots are often filtered by prerequisites like "you must have a 95% acceptance rate on your previous HITs"
- Traditional techniques for dealing with bad accuracy: best-of-3 or best-of-adaptive.

Our MTurk interface



Label named entities in Twitter data
Requester: [redacted] **Reward:** $1.00 per HIT **HITs Available:** 445 **Duration:** 10 minutes
Qualifications Required: HIT approval rate (%) is not less than 95

on the way to Tomales Bay for a BBQ w/ friends. discussing politi
tuned!

Word	Person	Place	Organization	None	???
on	○	○	○	●	☐
the	○	○	○	●	☐
way	○	○	○	●	☐
to	○	○	○	●	☐
Tomales	○	○	○	●	☐
Bay	○	○	○	●	☐
for	○	○	○	●	☐
a	○	○	○	●	☐
BBQ	○	○	○	●	☐
w/	○	○	○	●	☐
Word	Person	Place	Organization	None	???
friends.	○	○	○	●	☐
discussing	○	○	○	●	☐
politics	○	○	○	●	☐
and	○	○	○	●	☐

An entity is a object in the world like a place or person and a **named entity** is a phrase that uniquely refers to an object by its proper name (Hillary Clinton), acronym (IBM), nickname (Opra) or abbreviation (Minn.). Here are some more examples of named entities for each of the types we are interested in.

```
PER: Barack Obama; the Palins; John; ...
ORG: IBM; Coca-Cola Bottling Co., the Yankees; U.S.; ...
PLACE: Baltimore, MD; Washington; Mt. Everest; the Hoover dam; ...
```

When tagging named entities remember to:

- Tag words according to their **meaning** in the context of the tweet
- Only tag **names**, i.e. words that directly and uniquely refer to entities
- Only tag names of the types PER, ORG, and LOC

Running MTurk jobs

- Three jobs: 251, 1076, and 1755 HITs (each done twice).
- Fast: 15:04, 11:39, 20:20
- Cheap: $27.61, $118.36, $193.05
- Mostly okay: only a few empty-fills (i.e., answered all “none” when that wasn’t the correct answer), only one or two random-fills.

Best-of techniques

- Best-of-3
  - Run each HIT three times, then each annotation is a vote in a particular direction. Pick the most popular.
  - Can get 3-way ties if dealing with more than 2 possible answers, and it's not always clear which answer to take when dealing with floating-point or string-based answers.
- Best-of-adaptive
  - Run an initial batch with each HIT done three times, then re-run HITs with controversial answers until a concrete answer is found.
  - Perhaps break at some point to give an expert a chance to resolve the question, or give up and don't use as part of training data.

Using Gold data

- Popular idea: CrowdFlower combines this idea with adaptive best-of.
- Have experts generate some small amount of data, then ask workers the same question to see how they do.
- Our HIT includes this idea: one piece of gold data per HIT.
- How to judge workers? Look at how well they do on the gold questions (i.e., how close to the expert they come).
- A good idea, but it doesn't help you resolve conflicts very well with small numbers of workers: you could weight people's answers by how well they do on the gold, but gold isn't infallible.

An alternative: WorkerRank

- Based on PageRank (Brin and Page, 1998)
- Calculate the agreement between workers i and j for all i and j and store those values in $A_{i,j}$.
- Find the largest eigenvector x of A, and assign worker k the score x_k.
- Cheap to calculate the eigenvector, expensive to populate A fully, fairly cheap to update A on the fly.
- Many ways to think of this, but it can be thought of as reputation flow: if two workers agree on a particular annotation, they share some of their reputation with each other, but if they disagree they don't share.

A compromise

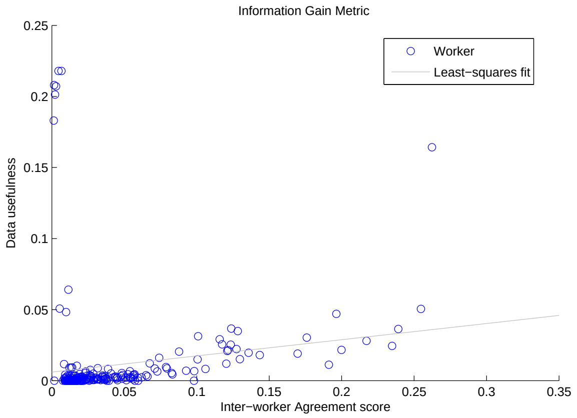
- Gold data is treated as being from a single worker, who's treated like every other worker.
- WorkerRank uses both the Gold data and the worker voting idea, in a way that lends itself to easy decisions about which data is good, and how much to trust the final data.
- The Gold data is treated as fallible, so you can find mistakes that were either made by the experts or consistently made by the workers.
- If the Gold data isn't being consistently agreed with, it'll be down with the rest of the workers in score, which means you may want to investigate your methodology or figure out if a small number of workers are causing trouble.

WorkerRank: comments

- A is by definition a real, symmetric matrix. Thus its eigenvectors are orthogonal and real. This means we can calculate the largest eigenvector using the power method and not worry about complex numbers.
- Runtime for this algorithm is $O(w^3 n + n^2)$ to initialize A, and $O(w^2 \mathrm{max\_iter})$ to calculate the eigenvector. The $w^3 n$ factor dominates, and in practice the time to initialize A is about two minutes, compared to a small fraction of a second to find the eigenvector.
- Workers who do more jobs earn higher scores, because they are sharing with a larger number of people. Thus, we normalize the columns of A so that they do not get so much undue influence.

Information gained

- Idea: How much does each worker contribute?
- Implementation: build one training set for each worker, then evaluate on gold data.
- Evaluate: how?
  - Accuracy: most answers are NONE—an algorithm that answers all NONE will get good accuracy, but it's not useful.
  - Precision: Again, mostly NONE, so getting a lot of those right isn't very interesting.
  - F_1 measure: harmonic mean of precision and recall. Precision is really noisy, because even small changes in the training corpus can change the number of NONEs, leading to big differences.
  - Recall: How many things that should be labeled non-NONE are?



Both axes are linear scales. Some workers get to label words that are in the training corpus and get much higher scores.

By the numbers

- We used the r^2 metric to check goodness of fit. This is also called the coefficient of determination.

$$r^2 \equiv 1 - \frac{SS_{\mathrm{err}}}{SS_{\mathrm{tot}}}$$

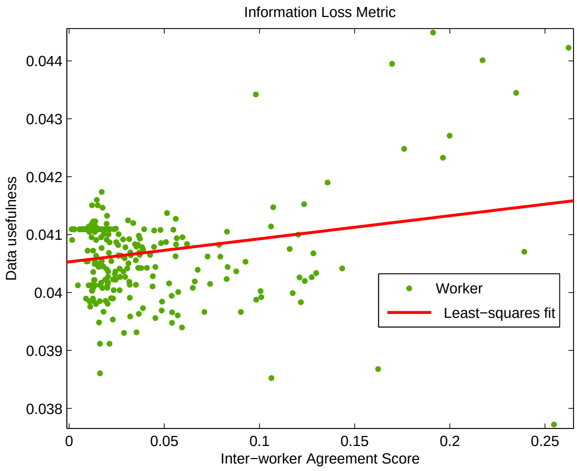
- (The name R^2 is given to the same goodness-of-fit measure when measuring non-linear fits.)
- The r^2 metric for this plot is 0.7280, and it's mostly driven up by the outliers; the fit is okay otherwise.

Information loss

- Idea: How much do we lose by losing the annotations from one worker?
- Evaluate the NER system trained on input from every worker but one.
- This is expected to be noisier, because there are many workers and the effect of taking one out will be small.
- Could be useful, though, because it's another data point.

How to build an info-loss training set

- Each tweet is annotated twice, so if we remove one worker's contribution, we end up with a bunch of things that we only have annotations for.
- If we just include these extra annotated-once tweets, we're giving more weight to the workers who annotated them, as well as removing the influence of the worker we take out. So this is a problem.
- But if a particular worker annotated a bunch of tweets that have many named entities in them, leaving the annotated-once tweets out will help their score a lot (because the training set gets worse when their annotations are taken out).
- It's a mess, but maybe we can get something out of it.

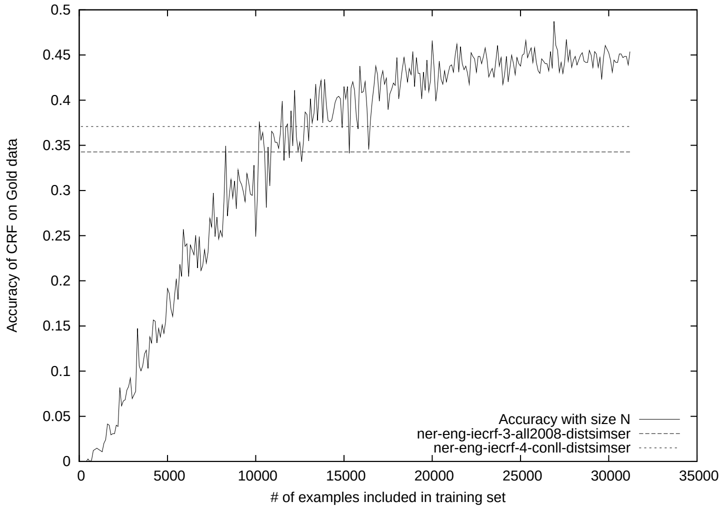

Information Loss Metric
Data usefulness
0.038
0.039
0.04
0.041
0.042
0.043
0.044
0
0.05
0.1
0.15
0.2
0.25
Inter−worker Agreement Score
Worker
Least−squares fit

- Bad fit: $r^2 \approx 0.5$.
- Conclusion: information loss isn't a very good metric. Worth a try, but not very good.

How much data is enough?

- We can easily test a baseline model against our Gold data.
- How many annotations do we need to include to beat that baseline?
- Train on randomly selected subsets of the annotations
- There are about 30,000 tweets annotated, we trained models whose size was a multiple of 100.

Evaluation Method for NER training data



AMT for NER

- As expected, it's cheap, it's fast, and the annotations it produces are noisy.
- Different schemes for cleaning up the noise have been proposed, but they often require many annotations or large amounts of Gold data.
- WorkerRank can save money by using more of the information we have at hand to better determine which workers are doing good work.

NER on AMT-generated training corpuses

- We can test WorkerRank's usefulness by comparing the workers' scores to the amount of information present in their annotation data.
- Not perfect: some words appear in Gold dataset, some entities show up twice for a particular worker, which means correct annotations can still result in no added information.
- Measuring workers' contributions by amount of information lost when they are removed from the training set is less successful: lots of noise, no good way to decide what to do with the other annotations of those tweets.

Questions

Any questions?