Improving Accuracy of Named Entity Recognition on Social Media Data

by

William Murnane

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
2010

ABSTRACT

Title of Thesis: Improving Accuracy of Named Entity Recognition on Social Media Data

William Murnane, Master of Science, 2010

Thesis directed by: Dr. Timothy W. Finin, Professor of Computer Science
Department of Computer Science and
Electrical Engineering

In recent years, social media outlets such as Twitter and Facebook have drawn attention from companies and researchers interested in detecting trends. The informal nature of status updates from these services leads to a higher volume of updates, because each update takes little care to generate, but each update is usually short and noisy (misspellings, lack of punctuation, non-standard abbreviations and capitalization). These shortcomings cause traditional Natural Language Processing (NLP) techniques to have substantially lower accuracy than is found with structured text such as newswire articles. We present a system for improving the accuracy of one NLP technique, Named Entity Recognition or NER, on Twitter data by training a recognizer specifically for this type of data. NER is the process of automatically recognizing which words are names of people, places, or organizations. This trained model is compared to baseline entity detection rate with an off-the-shelf NER system.

This thesis is dedicated to Donald Knuth, without whose software it would have been written on a typewriter.

ACKNOWLEDGMENTS

This paper would not have been started, let alone completed, without the influence of many people. Particular thanks are due to my advisor, Dr. Finin, who obtained the dataset which started the whole ball rolling and gave me invaluable advice along every step of the way, and my parents for their unwavering support (and proofreading skills). I'd also like to acknowledge the students in the eBiquity lab, who made useful suggestions about construction of auxiliary materials for this project—posters, slides, and so forth—and the administrative staff of the department. Finally, I should point out that this list is by no means exhaustive: many people over many years have conspired to elevate me to the position I enjoy today, and I thank them all.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Social Media has emerged of late as a new form of individual expression and communication. Services like Facebook and Twitter allow users to write short messages to each other and to the world at large, talking about current events, politics, products, or whatever comes to mind. These messages are generally very informal, commonly containing errors in spelling, improper punctuation, and a lack of capitalization, and are also quite short: for example, Twitter limits messages ("tweets") to 140 characters (Twitter support 2010), and Facebook limits status updates to 255 characters (Facebook.com 2010). They therefore pose a challenge to existing Natural Language Processing (NLP) techniques, trained on formal text like newswire articles or longer informal text like blog entries.

NLP techniques have been used in many domains, including finding structure in medicinal records (Friedman & Hripcsak 1999), using blogs to give political context to news articles (AAA 2008), and converting natural-language queries into machine-readable queries (Sun *et al.* 2007). These domains are often structured in some sense: they deal with a narrow range of topics, or take natural language input that is written by professionals. By contrast, we wish here to apply these techniques to text that is written off the cuff by average people in an informal setting.

A particular problem that we will discuss is that of Named Entity Recognition. This

technique addresses the problem of associating particular words with the entity (person, place, organization, or product, for example) that they represent. This process is language-dependent—for example, nouns in German are all capitalized (at least in formal contexts)—and we will only consider English to make the problem more manageable. Take for example this tweet from @radixextreme:

> Climate Change News: Bill Gates donates 20m to kickstart fund for farmers: Microsoft billionaire disappoint... http://tinyurl.com/255ansn

Here the word “Microsoft” represents the software company headquartered in Redmond, Washington, and the words “Bill Gates” represent its founder. There are two parts to this problem: determining which words represent an entity, and determining which entity those words talk about.

Examples of the second problem are easy to find. Microsoft is a simple example; there is only one corporation by that name. Consider the word “Columbia,” though: there are 21 cities across the US by that name, there was a Space Shuttle of that name, there is a film company called “Columbia Pictures” which is commonly abbreviated to simply “Columbia,” an American university called “Columbia University” which is also abbreviated “Columbia,” and many other examples. Considering the informal nature of the data we are dealing with, we might also choose to consider the problem of misspellings, and consider the South American country named “Colombia.”

This problem might seem insuperable, but using the context of a short message can quickly make it unambiguous which entity is being referenced. Consider this tweet from @MsArnold2U:

> @MJ_Dub lol sorry....i don’t think i will be home this summer...I’ll more than likely be in Columbia or St. Louis

This Twitter user's profile provides no hints; her location is set to "Somewhere Close to Success", which does not help. But using a knowledgebase we could look up all the senses of "Columbia" and "St. Louis" and find the pair that are closest in the hierarchy (or closest on the map, as we can probably guess that they refer to locations) and discover that the cities in Missouri are the most likely entities that these words refer to.

Chapter 2

LANGUAGE DETECTION

As mentioned in the Introduction, we wish to restrict our focus to English-language tweets only. However, the dataset we have is only rarely labeled with language (some 62,000 tweets of the 150 million), so it falls to us to discover the language of each tweet. The approach chosen comes from Maciej Cegłowski, the author of Perl's Language::Guess module, which is itself an implementation of the scheme proposed by Cavnar and Trenkle (Cavnar & Trenkle 1994). This algorithm was re-implemented in Java to give easier access to multi-threading facilities. It counts occurrences of n-grams (for n from 1 to 4) to build a profile of each language for which it has a sample, and then builds a profile (in the same way) of the text whose language is unknown, then applies a similarity measure between the profiles to determine which language is most similar to the unknown sample.

The samples used for this project were taken from Gertjan van Noord's TextCat program. These samples are not of any particular text, but there are many samples available, and they produce good results. Tweets were pre-processed to remove URLs and @-tags before performing language processing, but hashtags were included in the text (after removing their initial # character).

Out of the 60 languages that were detected, a few account for the vast majority of tweets. For example, 28% of the tweets were detected to be English (making it the most

English	42737702	Japanese	17848412	Scots	16279220
Catalan	6985391	German	6130202	Danish	4406936
Latin	3775001	Portugese	3482070	Spanish	3394897
Frisian	3353973	Dutch	2915385	Esperanto	2706490
French	2680402	Basque	2272255	Romansh	2115646
Slovak	1932566	Italian	1873306	Romanian	1775050
Indonesian	1756415	Nepali	1697859	Swahili	1590814
Swedish	1453701	Manx	1436424	Afrikaans	1338912
Breton	1278587	Tagalog	1199290	No language	1147953
Czech	955101	Finnish	922446	Farsi	794366

FIG. 2.1. Top 30 most common languages detected by the algorithm, and the number of tweets detected as being in each language.

common language), while the top three languages (English, Japanese, and Scots) accounted for 50% of the tweets. Figure 2.1 shows a more comprehensive picture of the languages detected by this algorithm. The presence of Scots as a language highlights one of the shortcomings of the algorithm: all languages are considered equally likely, and the model generated for Scots is similar enough to English that English text is often misdetected as Scots.

This approach has some definite advantages. This algorithm can recognize arbitrary languages given only a sufficiently long sample in that language; for example, our recognizer uses a corpus of some 60 languages to better differentiate English from non-English text. It can also generate a ranked list of which languages the unknown sample is most likely to be, which could be helpful to a human analyst; if she determines that the sample is not actually in the language most strongly suggested, the second choice is probably a good candidate for the actual language. In addition, it is fairly straightforward to give the recognizer more data on which to train. Once an input message is found that is labeled incorrectly, that message can go into the training file of the correct language, influencing the detection for future messages.

Some disadvantages of this algorithm are also worth noting. The algorithms for gen-

erating and comparing profiles are fairly slow, mostly because of the high cost in Java of garbage collecting all the n-gram strings generated in the profiling algorithm; with a little more care this could be avoided. This means that a single-threaded process can only detect the language of about 500 samples per second. This is mitigated by running multiple worker threads at once. In addition, the short nature of tweets means that there is not much to go on for any algorithm, so there is a relatively high probability of some spurious character combination causing the algorithm to misdetect the language of a sample. A few notable absences are also present. Chinese is poorly detected (and does not appear in the top 30 as a result), because this approach does not consider ranges of characters as hints. Therefore, in order for a character to convince the detector that the fragment is in Chinese, that character must appear in the source training text. An algorithm that considers ranges of characters might produce better results for Eastern languages.

Despite these problems, accuracy of the algorithm is quite good in the aspect that we most care about: its precision in finding English status messages. Recall is somewhat less good: many messages that are in fact English are marked as being in Scots, but the high precision produces a good result for this project. A thousand messages that were marked as English were selected at random, and checked by hand to see whether they were in fact in English. A total of only nine examples of non-English text were found, for a total precision of 99.1%.

How fast can data enter our system, and can we keep up? One website approximates the total flow from Twitter at around 1.2 billion tweets per month (Pingdom AB 2010). This is 40 million tweets per day, or 462 per second. Our current hardware is capable of running the language detection algorithm on 4000 tweets per second using 8 threads, which should be sufficient to keep up with occasional bursty behavior even if the entire Twitter stream were processed on a single machine. In the future it would be easy to parallelize this process across multiple machines if necessary.

Several improvements to this algorithm could be implemented to fit it better to this problem. For example, it is unlikely that a particular user tweets in a large variety of languages. Therefore, we could use knowledge of a user's prior tweets and what language they were detected to bias what language we determine the next tweet from that user is in. In other words, a user that has so far used only English and Spanish is unlikely to use Swahili or Arabic in their next tweet, so additional weight could be added to the already-used languages so that they are more likely to be detected. In addition, we could consider weighting on a global scale: for all Twitter users, Esperanto is rare, so the probability that a particular tweet is in Esperanto is low.

However, some difficulties arise in implementation of these ideas, which should be carefully considered before venturing too far down the path to them. This kind of weighting may place less weight on uncommon languages, lowering their probability of being chosen to the point that only common languages are ever detected. Unless weights are chosen carefully, an unweighted detector might do a better job finding tweets in these uncommon languages. The global weighting must be implemented carefully, too, so that the order tweets are processed in does not yield too large a change in the results that are obtained. For example, we might initially see a large number of tweets in Dutch as a result of some local event, conclude that Dutch is a very common language on Twitter, and weight it highly. Later analysis might show that this is not the case, but until the weight on Dutch is reduced the number of tweets misdetected as Dutch would be elevated, slowing the reduction of the bias toward that relatively uncommon language.

Chapter 3

DATASET DESCRIPTION

For the purposes of improving entity recognition, there is no substitute for experimentation on real data. The dataset used for this paper consists of Tweets collected by a third party over 20 months, from March 2007 until November 2008. The distribution of the number of tweets collected per day is presented in Figure 3.1; we note that the collection method appeared to change in the Spring of 2008 from a maximum of approximately one per second (i.e., 86,400 per day) to retrieving at a much higher rate. The data are comprised of SQL tables, stored in the PostgreSQL database engine (Group 2010). There are approximately 150 million status updates, written by 1.5 million users. Each user has an associated user record, which tracks information like their Twitter ID number, their account creation time, and other things that Twitter stores per user. Users are also associated via a foreign key with a location; locations are stored in normalized form in another table.

The tweets are all in plaintext, but there are some conventions—hashtags, user mentions, retweets, and URLs—that give additional metadata about the contents of the tweet. Hashtags are words or abbreviations prefaced with a "#" character, which are used either as metadata or as a word. User mentions consist of a username prceded by an @ character, and can be used to name a person or to direct a tweet at them. Retweets are designated by the characters "RT", followed by a username and the text of a tweet posted by that user.

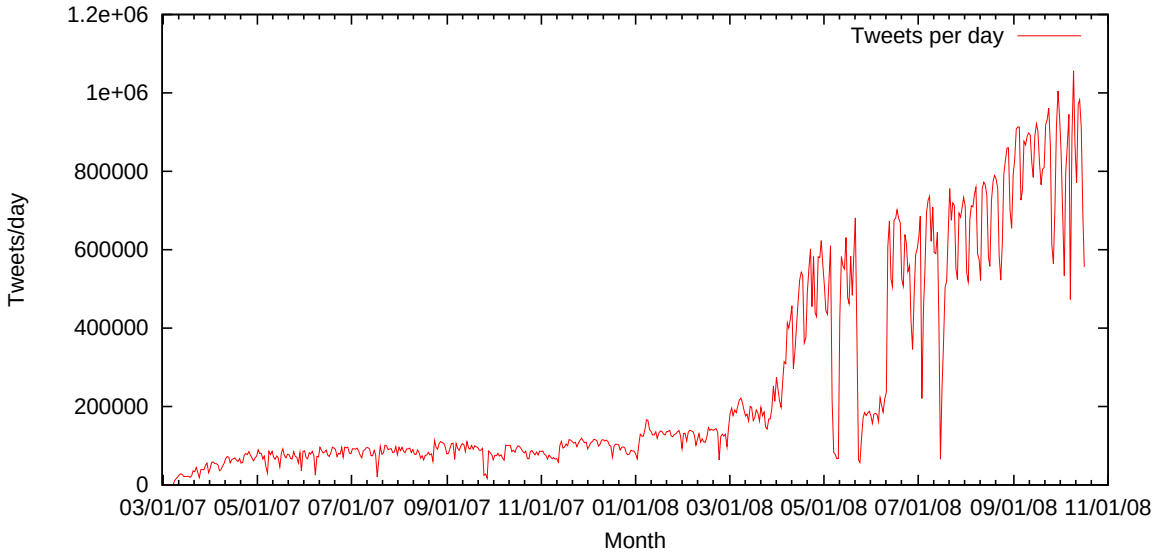

FIG. 3.1. Number of tweets captured per day for the entire dataset.

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`

Table 3.1. Examples of Twitter conventions. *Emphasized* words or phrases highlight each convention.

URLs are also often present in tweets, using the familiar `http://` notation, but often use an external URL shortener like `bit.ly` or `is.gd` to better fit within the 140 character limit of Twitter (Mateosian 2009). Examples of all of these conventions are demonstrated in Table 3.1. Information about individual status messages are also stored in separate tables for fast lookup; hash tags, username mentions, and URLs are all kept in tables and associated many-to-many with status messages.

In addition to the SQL database, a text-search index of the tweets is maintained, using the Lucene library. This Lucene index allows for fast searches for tweets containing particular words. Only those tweets whose language is detected as English (see Chapter 2) are kept in the index, because doing NER in non-English languages is beyond the scope of this work and this limitation keeps the size of the index manageable. This language detection is also used to select English tweets for further processing.

The tweets are stored (more or less) in order of time when sorted by primary key. In order to get a good training set, we added a field to each tweet called `rand_order`. This contained an integer between 0 and the size of the dataset, in a pseudo-random order. Ordering the dataset by this random field and selecting consecutive entries gives a random subset that does not overlap with previous selections.

Chapter 4

AMAZON MECHANICAL TURK

Improving the NER task we wish to tackle takes a lot of training data, which should be generated by hand. Toward this end, we made use of the Mechanical Turk service provided by Amazon.com. The service is described on its FAQ page as

> a marketplace for work that requires human intelligence. The Mechanical Turk service gives businesses access to a diverse, on-demand, scalable workforce and gives workers a selection of thousands of tasks to complete whenever it's convenient.
>
> Amazon Mechanical Turk is based on the idea that there are still many things that human beings can do much more effectively than computers, such as identifying objects in a photo or video, performing data de-duplication, transcribing audio recordings, or researching data details. Traditionally, tasks like this have been accomplished by hiring a large temporary workforce (which is time consuming, expensive, and difficult to scale) or have gone undone. (Amazon.com, Inc. 2010)

The service provides a convenient computerized interface for Requesters (people or organizations who want work done) to upload descriptions of Human Intelligence Tasks (often abbreviated to HITs). Then workers can complete these tasks and earn a reward. Amazon

charges requesters a fee for finding them workers to do their jobs: half a cent or ten percent of the reward offered to the workers, whichever is greater. The service is often abbreviated MTurk.

For our task, the interesting question is "which of the words in these tweets represent named entities?" This is not entirely determined by the words themselves, but at least partially by their positions: "The" and "Who" are not named entities, but "The Who" certainly is. A modern machine learning technique for this type of sequence tagging is linear chain conditional random fields, or CRFs (Lafferty, McCallum, & Pereira 2001). We will not delve into the implementation of CRFs (although Chapter 5 does give more details), because our goal here is not a general-case NER system, but an improvement in a specific genre of text.

Expert humans can do a good job at this task, but we are asking non-experts to do our tagging. We expect that English speakers will be able to do some sort of a reasonable job on this task, but with some amount of disagreement on subtle points. This is not so bad for an entity recognizer; the trainer will put less faith in things that have conflicting answers, so we could get fairly good answers by simply training on all the human answers that we receive. We will still want to entirely disregard answers that are of poor quality, though, including those generated by machine and those from non-English speakers.

With this in mind, the thing we must consider is discovering workers who are acting in bad faith: using software to perform a task rather than do it by hand, for example. We can do this by building a small amount of "Gold" data by hand (so that we know it is done by humans) and including one tweet from that dataset in each HIT. Then we can use inter-worker agreement to judge how much workers agree with the known human judgements.

The mixing algorithm has a few concerns; first, it should be as unpredictable as possible, to prevent an adversary from determining which tweet has the Gold answer that they will be judged strongly on and spending time on that one tweet at the expense of the others.

```
MIX : gold, unlabeled → mixed
1  goldleft ← NEW-SET(gold)
2  while | unlabeled | > 0
3       do newHIT ← RANDOM-SAMPLE(unlabeled, 4)
4          if | goldleft | = 0 then SET-ADD-ALL(goldleft, gold)
5          newHIT[5] ← RANDOM-SAMPLE(goldleft, 1)
6          SET-REMOVE(newHIT[5], goldleft)
7          RANDOM-SCRAMBLE(newHIT)
8          ▷ Write newHIT to the output.
```

FIG. 4.1. Mixing algorithm. This takes as input a list of Gold tweets and a list of unlabeled ones, and outputs HITs which have one Gold and four unlabeled tweets, in a random order.

Second, it should be frugal with the Gold data, putting the minimum amount in each HIT necessary to find an accurate picture of inter-worker agreement. Figure 4.1 shows the algorithm developed to mix Gold tweets in with unlabeled ones. Maintaining the set of gold which has been used spreads the gold as far as possible.

Having thus mixed the gold data with the un-annotated data, we next consider what sort of interface to give the workers to allow them to annotate the tweets. We decided on a narrow vertical interface rather than a wide horizontal one so that workers could scroll vertically rather than horizontally. This vertical, one-word column layout hurts readability of the tweets somewhat, so we provided a horizontal version of the tweet above each set of columns. Each tweet was presented in tabular form, with each row containing a single term from the tweet, a set of radio buttons for indicating its type, and a checkbox to let the worker indicate that they were unsure about their annotation. The row of headers which describes the columns was repeated every 10 rows, so that workers would not have to scroll back and forth to see the headers. Finally, the word “Help” in red was displayed in the upper-left corner of the window (regardless of scroll position). Whenever a worker hovered their mouse pointer over this word, our annotation guidelines were displayed. Figure 4.2 shows

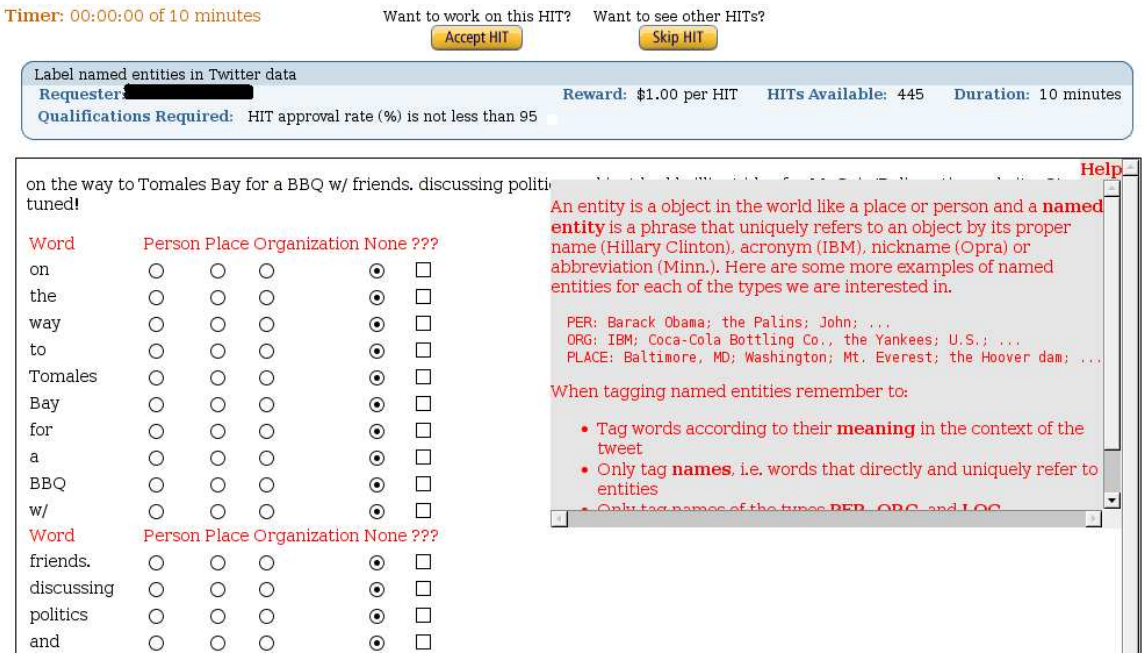

FIG. 4.2. Amazon MTurk interface for workers.

the HTML interface, and Figure 4.3 shows the text of the annotation guidelines we gave.

We ran three batches of HITs with this template, which included 251, 1076, and 1755 HITs in order. Each HIT was completed by two workers, included five tweets, and gave a reward of $.05. Thus, a total of 6160 HITs (counting the twofold expansion) were completed, containing 12320 previously unannotated tweets and 3080 Gold tweets.

To do the actual inter-worker comparisons, we present WorkerRank, an algorithm based on Google's PageRank algorithm (Brin & Page 1998). Figure 4.4 presents pseudocode for the algorithm. One function is left undefined: the SIMILIARITY function on line 4. This function is straightforward: it looks at all the HITs that both workers did (and if there are none, returns 0) and compares them using a second function called AGREE, which is defined in Figure 4.5.

The algorithm in Figure 4.4 has several steps, runtimes of which can be analyzed independently. Initializing the matrix A calls SIMILARITY $\Theta(w^2)$ times, where w is the

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; ...
```

Pronouns (me, I, we, they) should not be tagged, but Twitter usernames like @barackobama should be tagged.
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*
- You can check the *???* box to indicate somthing you consider to be ambiguous or that you are uncertain about

FIG. 4.3. Annotation guidelines for Mechanical Turk workers.

WORKERRANK : $results \rightarrow scores$

1 $worker\_ids \leftarrow$ ENUMERATE(KEYS($results$))
▷ Initialize A
2 **for** $worker1 \in worker\_ids$
3 **do for** $worker2 \in worker\_ids$
4 **do** $A[worker1, worker2] \leftarrow$ SIMILARITY($results[worker1], results[worker2]$)
▷ Normalize columns of A so that they sum to 1 (elided)
▷ Initialize x to be normal: each worker is initially trusted equally.
5 $x \leftarrow \left\langle \frac{1}{\sqrt{n}}, \ldots, \frac{1}{\sqrt{n}} \right\rangle$
▷ Find the largest eigenvector of A, which corresponds to the agreement-with-group value for each worker.
6 $i \leftarrow 0$
7 **while** $i < max\_iter$
8 **do** $x_{\text{new}} \leftarrow$ NORMALIZE($A \times x$)
9 $diff \leftarrow x_{\text{new}} - x$
10 $x = x_{\text{new}}$
11 **if** $diff < tolerance$
12 **then break**
13 $i \leftarrow i + 1$
14 **for** $workerID, workerNum \in worker\_ids$
15 **do** $scores[workerID] \leftarrow x[workerNum]$
16 **return** $scores$

FIG. 4.4. Intra-worker agreement algorithm. MTurk results are stored in an associative array, with worker IDs as keys and lists of HIT results as values, and worker scores are floating point values. Worker IDs are mapped to integers to allow standard matrix notation. The Similarity function in line 4 just returns the sum over HITs done by two workers of the AGREE function on the tweets whose IDs match.

AGREE : $words, labels1, labels2, unsure1, unsure2 \rightarrow score$

```
1   agree ← 0.0; disagree ← 0.0
2   for i ∈ [0, | words |)
3       do if labels1[i] = labels2[i] ∧ ¬ unsure1[i] ∧ ¬ unsure2[i]
4             then agree ← agree +1.0 ▷ These annotators agree completely.
5     else  if labels1[i] = labels2[i]
6             then agree ← agree +0.7
7                  disagree ← disagree +0.3
                   ▷ These annotators agree, but one or both are unsure, and we
                     do not want to reward that as strongly.
8     else  if ¬ unsure1[i] ∨ ¬ unsure2[i]
9             then agree ← agree +0.5
10                 disagree ← disagree +0.5
                   ▷ These annotators disagree, but they reported being uncertain, so
                     they will not lose as many points.
11    else  disagree ← disagree +1.0
12  return agree /(agree + disagree)
```

FIG. 4.5. AGREE compares two tweets, and assigns them a value between 0 and 1 representing how much their annotators agree about how to annotate them.

number of workers. Normalizing the columns of A is a $\Theta(w^2)$ process, and initializing x is $\Theta(w)$. Then the power series calculation in lines 7–13 takes $O(max\_iter\, w^2)$. SIMILARITY does a hash intersection of the HITs that each worker has completed, taking $O(w_1 + w_2)$ (where w_i is the number of HITs that worker #i completed) to find the set of HITs that both workers have completed; it then calls AGREE some number of times, but we will consider that separately. AGREE itself takes a constant amount of time to run, since there are only at most 140 characters (and thus 70 labels, with one-character words) in either tweet to compare.

How many times will AGREE be called? If a particular tweet is annotated k times, then for each pair of workers that have annotated that tweet, AGREE will be called once. Thus, for each tweet, AGREE is called $O(k^2)$ times. For our dataset, most tweets are annotated exactly twice, but for those tweets in our "Gold" dataset, we will have many annotations: one from us, and one for each time that tweet appears in a HIT. Because the mixing process tries to avoid repeating Gold data, the maximum number of times a Gold tweet will be part of a HIT is $O((n/4)/g)$, where n is the number of tweets that are to be annotated and g is the number of Gold tweets. Since each HIT is annotated twice, the total number of times a Gold tweet is annotated is $2\,O((n/4)/g) + 1$. So the total time spent calling AGREE on Gold tweets is

$$O\left(\left(\frac{n/4}{g}\right)^2\right).$$

The asymptotic bound is not tight here; the actual value may in fact be smaller than this, because a worker may be randomly assigned two tweets which share the same piece of Gold data, although they are prevented from doing the same HIT twice. In addition, each non-Gold tweet is compared twice (once when comparing the first worker who did that tweet to the second, and once when comparing the second to the first) for a total of $O(n^2)$ comparisons.

Now we almost have enough information to assemble an overall asymptotic runtime analysis. The only remaining puzzle piece is how long the hash intersection in SIMILARITY takes. We hedged earlier and said that this takes $O(w_1 + w_2)$ time per call, but we can get a better answer by considering that we will eventually calculate this intersection for all pairs of workers. Since we know how many HITs there are ($n/4$), and we see that the sum doesn't depend on the product of w_1 and w_2, we can evaluate the sum:

$$\begin{aligned}
\sum_{i \in workers} \sum_{j \in workers} w_i + w_j &= w \sum_{i \in workers} w_i + w \sum_{j \in workers} w_j \\
&= w \left(\sum_{i \in workers} w_i + \sum_{j \in workers} w_j \right) \\
&= 2w \left(\sum_{i \in workers} w_i \right) \\
&= 2w(n/4) = wn/2.
\end{aligned}$$

In summary, the algorithm described takes $O(w^3 n + n^2)$ to initialize A, and $O(w^2\, max\_iter)$ to find the eigenvector using the power method. In practice, with $max\_iter = 50$, a tolerance of 10^{-6}, $w = 270$, $g = 441$, and $n = 6160$, the algorithm takes about 114 seconds to initialize A and .03 seconds to calculate the eigenvector (taking 47 iterations).

Chapter 5

NAMED ENTITY RECOGNITION USING CONDITIONAL RANDOM FIELDS

While this agreement algorithm looks interesting, evaluation of the quality of this data on the real task must be the final arbiter of usefulness. We will use the Stanford NER tools to train an NER system and then evaluate worker effectiveness in an actual setting.

Other NER systems were considered, including the one included with the Mallet language toolkit and the one included with the LingPipe suite of libraries. Mallet was rejected because it requires manually building features, and discovering interesting features for Twitter statuses is outside the scope of this work. The Stanford NER tools include many standard feature extractors, which makes it simple to compose a system that is both easy to use and easy to replicate: if we wrote our own feature extractors, other implementations might differ on subtle points, but if we use the off-the-shelf feature extractors it is easier for others to replicate our results. LingPipe was rejected because it uses a Hidden Markov Model, a less flexible machine learning technique.

5.1 Algorithm overview

The algorithm underlying the Stanford software is based on the idea of Conditional Random Fields, or CRF, originally described by Lafferty, McCallum, & Pereira (2001). As

the goal of this project is not to provide a better algorithm but better training data for the existing CRF-based algorithm, we will provide only a brief overview of the algorithm. A more complete description of CRF is provided in (Sutton & McCallum 2006).

Using probability networks for machine learning requires in some sense guessing ahead of time which things are caused or biased by other things. A convenient way of representing this network is a graph where nodes represent inputs and outputs, and edges their influence on each other. Different methods can be represented as graphs; the edges represent an influence of an input on an output or another input, with an edge weight representing the amount of influence that input has on that output.

A linear-chain CRF uses this idea of dependency. Here the inputs are fully connected to the outputs, but the inputs are also connected to each other in an acyclic manner: a linear chain. This is a restriction on the shape of the graph, but it has an advantage in terms of calculability: the number of parameters that are estimated in training is smaller than in the general case, which means training can be substantially faster. The Stanford NER system uses a modification of the linear-chain CRF in which observations are clustered into cliques of pairs of states (Finkel, Grenager, & Manning 2005). Probabilistic transition values between members of each of these cliques are used rather than full probability distributions, which makes calculating the most likely state sequence less expensive.

5.2 Evaluation Method: Information Gain

We start with a simple method of evaluation: how much information do we gain from a single worker's annotations? We will train several NER models, each on the subset of tweets annotated by a particular worker. Then we will investigate correlation between the inter-worker agreement score and the gain in accuracy of the NER system in annotating the gold data.

There are many fewer considerations we must make here, since our dataset consists only of a single worker (so we need not control for the effects of other workers). We still need to control for the number of HITs a worker completes, though, since a bigger training dataset will generally produce better results, and we are interested in the quality of individual annotations that workers produce.

For this purpose we shouldn't use a linear division, because annotating a large number of HITs means that the incremental information about worker quality gained from a single tweet goes down. That is to say, if worker performance is consistent, we will have a mostly-correct impression of how good a job the worker is doing after only a few HITs, and from then on we will gain less and less information about their quality. So, to counteract this factor, we will divide the workers' performance on the gold data by a logarithmic factor:

$$l_{j,\text{overall}} = l_j / \log(w_j).$$

This corrects for the fact that a bigger training corpus will produce better results without asking too much of workers.

We define a function NER-EVAL, which takes a set of training data, trains an NER system on it, and then tests that system against the gold-standard data. This function needs to test the proper metric. Accuracy (the number of correct answers) is a poor choice, because most words do not refer to entities. Because of this factor, the precision (how many of the annotations bear the correct label) is also somewhat uninteresting: labeling every word "none" will give a high precision score, while providing no information. Even the F_1 measure (which takes into account both precision and recall) is not useful, because of the large degree of noise in the precision scores. Instead, we will evaluate based on recall: of the words that should have a non-"none" label, how many have the correct label?

An alternative method of measuring usefulness of data is to consider each type of

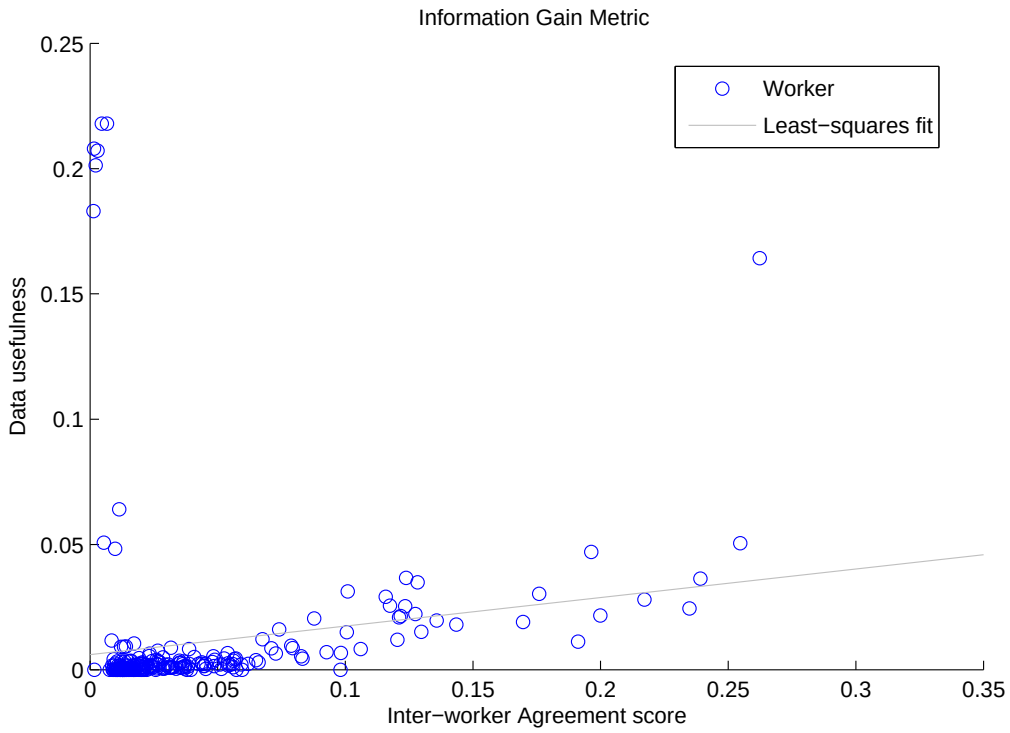

FIG. 5.1. Inter-worker Agreement score versus information gain from each worker.

entity independently. The recall measure is a reasonable one, but we might be able to pick more interesting trends out of the data if we look at precision and recall on, say, just entities whose correct label is “person”. Appendix A discusses this strategy, and its results, in more detail.

NER-EVAL is implemented, then, by training an NER system on a corpus composed of the results of a single worker, not using any other workers as a check of correctness. This is done by taking each worker entirely at their word: no matter how many people think they are wrong, all of their annotations and only their annotations are included in the training corpus. Then we check their recall on our gold training data.

Our results are fairly good, as shown in Figure 5.1. The adjusted r^2 metric (a measure of goodness-of-fit) has a value of 0.7280, driven down mostly by the few outliers. A better

fit might be interesting, but the two things that are being compared against each other are both indirect measures. The data that is collected from a worker may be good (that is, correct annotations), but be misleading nevertheless because the word usage in the sample of text is unusual. Consider a mention of "The Who" which is labeled as an organization; this might cause further mentions of "who" to be labeled as an organization, reducing the precision of the output. Despite the fact that the output of this model gets worse, it is still based on a larger amount of correctly annotated data than if "The Who" were labeled as non-entities.

5.3 Evaluation Method: Information Loss

Another method of using an NER system to evaluate quality of workers is to consider subsets of the training data to which particular worker did not contribute against all of the training data. Formally, we give the label w_i to the work done by worker i; then we will calculate the information loss metric for worker j (which we label l_j) as

$$l_j = \textsc{Ner-Eval}\left(\sum_i w_i\right) - \textsc{Ner-Eval}\left(\sum_{i \neq j} w_i\right).$$

One problem we face with this scheme is that when we remove a worker, all of the tweets that worker annotated were annotated by exactly one other worker (except the gold-standard data), because each set of tweets was done by exactly two workers. Then when we attempt to build a training set that excludes this worker, we are left with a number of tweets that are annotated by only one worker (for which we cannot calculate agreement). If we were to include these in the training set, we would be trusting those workers unduly much: their answers would be included regardless of whether they agree with another worker. This would lead to an evaluation that included workers that worked with worker j being weighed more heavily compared to other workers. Because the effects of this are not really random

(the worked-with relation is not randomized each time the testing is run, but determined when the work is done) it would be problematic to exclude this effect.

On the other hand, if we exclude these tweets entirely, we are reducing the size of the training corpus by the total length of the jobs the worker did. So we have to account for this somehow. We might anticipate that the amount of data contained in a training corpus is proportional to some function of the number of examples in it. So when we calculate the amount of information lost for a particular worker, we also want to weight the size of the remaining corpus. That is, for the training set t_j composed of answers from all workers except worker j and people who did the same jobs as her, we will calculate l_j, the amount of information lost by excluding worker j from the training corpus. Then we must account for the fact that we are working with a smaller input corpus by multiplying by the ratio of the size of t (the training set which includes all the workers) to the size of t_j. To compensate for the fact that adding training examples improves the evaluation function only slowly, we will multiply by the quotient of logarithms:

$$l_{j,\text{overall}} = l_j * \frac{\log |t|}{\log |t_j|}.$$

The final effect we might want to consider is how many tweets each worker annotated. The last compensation we made controls for how many words the workers annotated, but not all tweets are the same number of words. We might try to compensate for this by adding a factor that we multiply each l_j value by, but the problem is that the number of words a worker annotates and the number of tweets they annotate is strongly correlated, so compensating for both things independently would overcompensate. Instead we will treat them as mostly dependent and check that this is the case. Figure 5.2 shows the number of HITs worker j completed (on the horizontal axis) against the number of words in t_j. A strong correlation between number of words and number of HITs completed (adjusted r^2

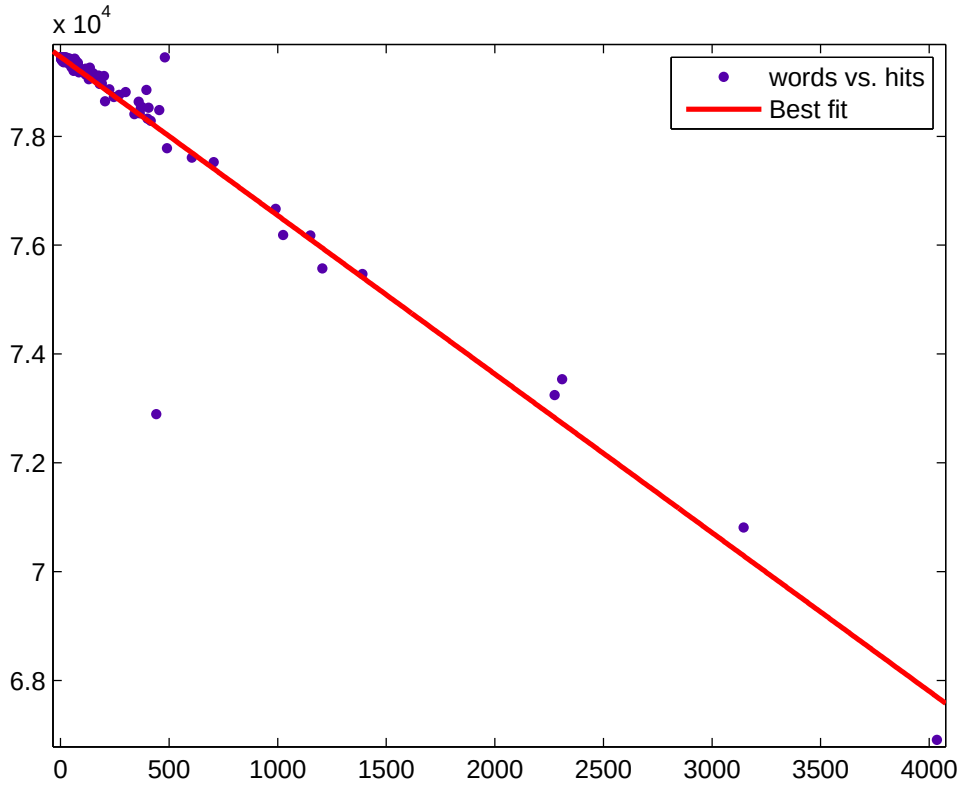

FIG. 5.2. Number of HITs worker j did (x-axis) versus number of words in the training set t_j. Each point represents one worker, and the line is a sum-of-squares best fit.

metric: 0.9181) shows that we need not compensate for the number of tweets and words independently: a single correction is nearly enough, and correcting twice would be overkill. Therefore we will make no additional correction, and the previous equation will be used.

Now that we have settled how to do the information-loss evaluation, we can actually run the evaluation. We trained w entity recognizers, each excluding one of the workers, and labeled them with the worker who was excluded. In other words, they were trained on the t_js introduced in this section, and labeled with j. Then we evaluated each recognizer on the whole input data: the training set that the workers generated. We also trained a single recognizer on the whole training set t as a baseline. These machine-generated annotations

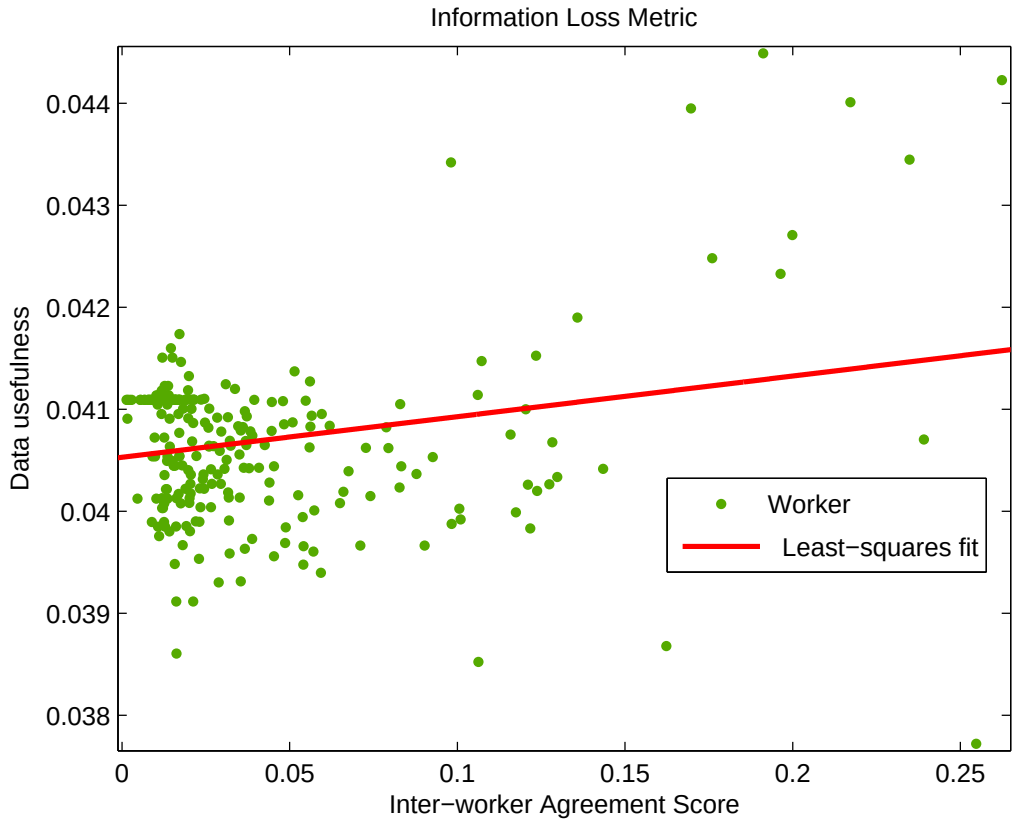

FIG. 5.3. Evaluation of information lost by eliminating worker j versus score of worker j. The line is a sum-of-squares best fit.

were compared to the human-generated ones to find evaluation values e and e_j. Then we used the following formula to calculate the actual per-worker information loss associated with leaving that worker out of the training set:

$$l_{j,\text{overall}} = (e_j - e) * \frac{\log |t|}{\log |t_j|}.$$

Results from this test are shown in Figure 5.3. The fit is quite bad; the adjusted r^2 metric is 0.05003, and the 95% confidence interval is large: the coefficient of x in the equation of the line is estimated to be between 0.00009 and 0.0002.

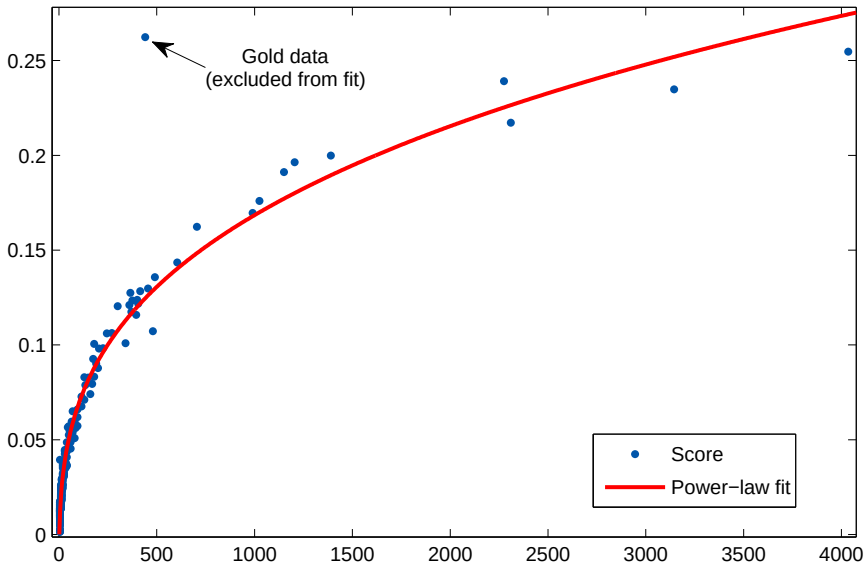

FIG. 5.4. Scores awarded to workers, by number of HITs completed. The fit line is $0.2284x^{0.3106} - 0.2681$, and the adjusted R^2 measure is 0.9854.

Why does this data fit so poorly to a straight line, as compared to the fairly good straight-line fit we saw in the previous section? One hypothesis is that removing all the HITs that are annotated by only one worker when ignoring annotations from worker j causes enough noise that the correlation is lost. Another is that the logarithmic fit we're using is an incorrect assumption. This could cause problems because the inter-worker agreement score is correlated with number of tasks done. Figure 5.4 shows the scores workers who completed a given number of HITs earned. We might expect to see some correlation between the scores earned by workers and the number of HITs they complete, but we might hope it would not be so blatant. However, the fact that this power law fits the plot so well, rather than a linear or quadratic function, seems to imply that this is not a matter of a missing factor somewhere in the agreement algorithm, but rather that this correlation exists in the data.

Chapter 6

RELATED WORK

Inter-annotator agreement is a subject which is studied in the context of many different Mechanical Turk tasks. Almost every task that is to be performed will be done by more than one worker, so for each task a new metric of agreement must be derived. Expert-generated data is also often used to better measure worker quality.

In (Jiang & Zhai 2006), the authors explore the possibility of creating cross-domain models for NER by looking for general features that work well across multiple domains, and focusing on those. They show a slight improvement over a baseline technique when training on one genre of text and testing on another.

(Snow *et al.* 2008) investigates the quality of expert versus non-expert annotations for five natural-language tasks, and compares experts to varying numbers of non-expert annotators. Their voting mechanism is similar to one of the baseline schemes we used.

(Locke & Martin 2009) investigates this topic using a Support Vector Machine approach on a substantially smaller dataset. Performance of this approach is 54% accurate for locations, approximately 30% for organizations, and 18% for people. An approach that trained a model on pre-annotated Twitter data (much like the approach examined in this paper) achieves an F1 measure of 59.5% overall.

Chapter 7

CONCLUSION

We built a platform for managing the large volume and high flow rate of social media data, including database design, building a sufficiently fast language detection algorithm, and creating extraction tools to quickly retrieve a subset of our tweet collection for further processing. A template for Amazon's Mechanical Turk service was also developed, allowing for quick and inexpensive collection of training data for a machine learning algorithm to perform named entity recognition.

In order to better focus our efforts on English-language Tweets only, it was necessary to determine which language the Tweets in our collection were in. Toward this end, a language recognition algorithm was developed and used to determine the language of each tweet. Results from this process were good, acheiving better than 99% precision. Precision on other languages, and recall on any language, were not investigated.

Building on this foundation, we have developed an algorithm for finding inter-worker agreement, and two approaches to evaluating performance in a different but related domain: quality of annotation data. The first metric, based on information gained from a particular worker's annotations, shows that the inter-worker agreement metric correlates well to the amount of information that a single worker contributes. The second, based on information loss associated with ignoring a particular worker's annotations, does not correlate at all,

perhaps because of confounding factors. This WorkerRank algorithm provides a simple method of finding workers who are producing high-quality annotation data.

This process provides an inexpensive way to produce a training corpus for machine learning algorithms, using untrained workers and only a small number of seed gold answers. From an expert perspective, this could be thought of as a semisupervised learning algorithm: the expert produces a small amount of training data, then uses that data to generate a larger corpus with the assistance of inexpensive crowdsourcing services like Amazon Mechanical Turk. This larger corpus has been obtained without additional expert intervention, but provides a significant improvement in quality over the expert data alone.

Many directions for future work arise from this point forward. Improvements could be made to the language detection algorithm, either in using a more modern approach or providing genre-specific training data. A different algorithm of evaluating worker quality could be developed and compared to WorkerRank, using it as a baseline. A framework for collecting tweets and recognizing entities using a model could be developed.

Appendix A

ALTERNATIVE METHOD OF EVALUATING NAMED ENTITY RECOGNIZERS

In Chapter 5, we compared efficacy of several models derived from subsets of our AMT-derived training data to WorkerRank to attempt to determine whether a particular worker's good WorkerRank score is a good indicator of high-quality training data coming from that worker. Here we revisit this topic, with a different scoring metric for model efficacy.

The metric discussed in Chapter 5, called there "recall", evaluates based on how many of the things that should have a non-"none" label end up with the correct label. For the sake of less ambiguous discussion, we will call this metric "simple recall". This is a useful metric, but it may hide some useful subtleties. For example, suppose that taking some existing model and adding training data from some worker causes the model's performance to go down in predicting person entities, but increases its performance on organization entities. The simple recall metric will show that this worker has had an overall zero effect on the quality of the model produced, while really there are both positive and negative effects of adding this worker. If we care more about finding organization entities than people, we might wish to include this worker's contribution, despite the fact that this will hurt the model's performance on "person" labels. A slightly more complicated metric can

$$\begin{array}{cc} & \text{Label from model} \\ \text{Correct label} & \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \end{array}$$

FIG. A.1. A small example confusion matrix.

show us that this effect is happening.

We can easily build a confusion matrix for this problem: how many times was there a token that should have been labeled x, but was labeled y? We will label the rows as "should have been labeled x", and the columns as "was labeled by this model as x", as shown in Figure A.1. A perfect model will have a diagonal confusion matrix: whenever a thing should have been labeled x, it was labeled x. A model that produces slightly incorrect results will produce a confusion matrix strongly weighted toward the diagonal, and a model that predicts by random choice will have columns whose expected sum is the same, and rows whose components are expected to be proportional to the distribution of the correct labels.

Using this confusion matrix, we can come up with precision and recall numbers per label. In other words, we can answer questions like "of those entities that should have been labeled 'person', what fraction of them got that label?". For this example, we will find this number by summing across the row labeled "person", and then dividing the entry "person, person" by the row sum. This number will be less than or equal to 1, since the entries of the matrix are positive and the row sum includes the single element in the numerator. Similarly, we can find the fraction of entries who were labeled "person" which should have had that label by summing down the "person" column and dividing "person, person" by that column sum. These two numbers are the recall and precision, respectively, of the model for the label "person". We will call this type of metric the "person recall" or "person precision"

of a model, or more generally the “complex recall” or “complex precision”.

Presented below are a large number of graphs. First we have the positive evaluation method: use the answers from one worker to construct a model, then evaluate the goodness of that model with one of the complex metrics and compare that to the worker’s WorkerRank score. Next is negative evaluation, in which models based on subsets of the MTurk results which exclude one worker are built and compared to the WorkerRank of that worker. Finally, we include a new type of graph, based on choosing random subsets of size n of the results from MTurk. Here the goal is to see how much data we should collect before concluding that our model is sufficiently good (or at least, as good as it is going to get). These random subsets are chosen independently: there is no guarantee that a subset of size $k + \epsilon$ will be a superset of the subset of size k. This third evaluation method can show how many answers should be gathered to train a good model; as long as the precision or recall for the desired label increases, it might be worth trying to collect more data.

Finding a straight-line correlation in the first two sets of graphs would be nice, but as discussed in Chapter 5, such a correlation would be between two things that should not necessarily be related, in that both things are indirectly related to the actual quality of the data.

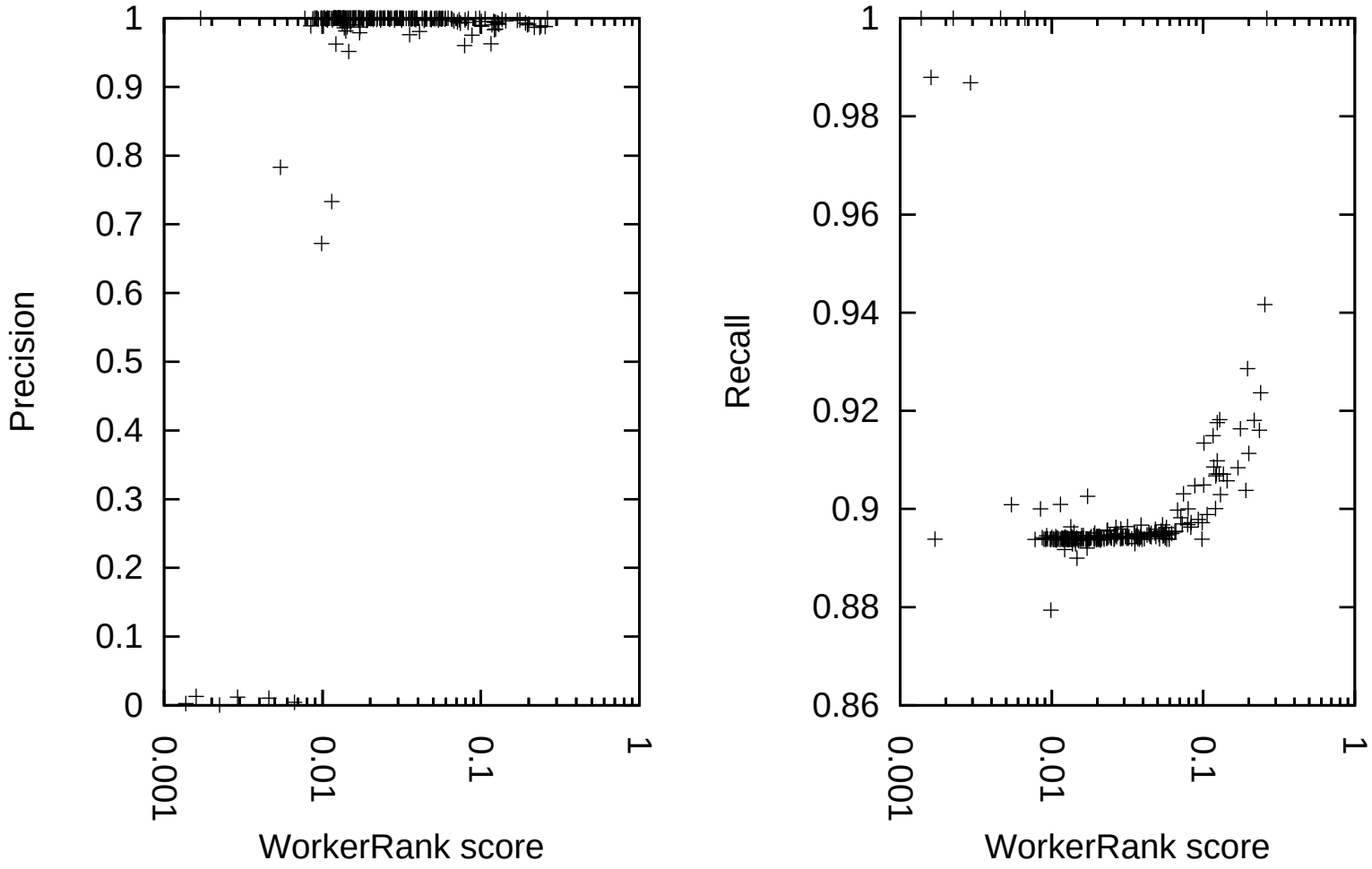

FIG. A.2. Precision and recall for “none” term, for the positive evaluation method.

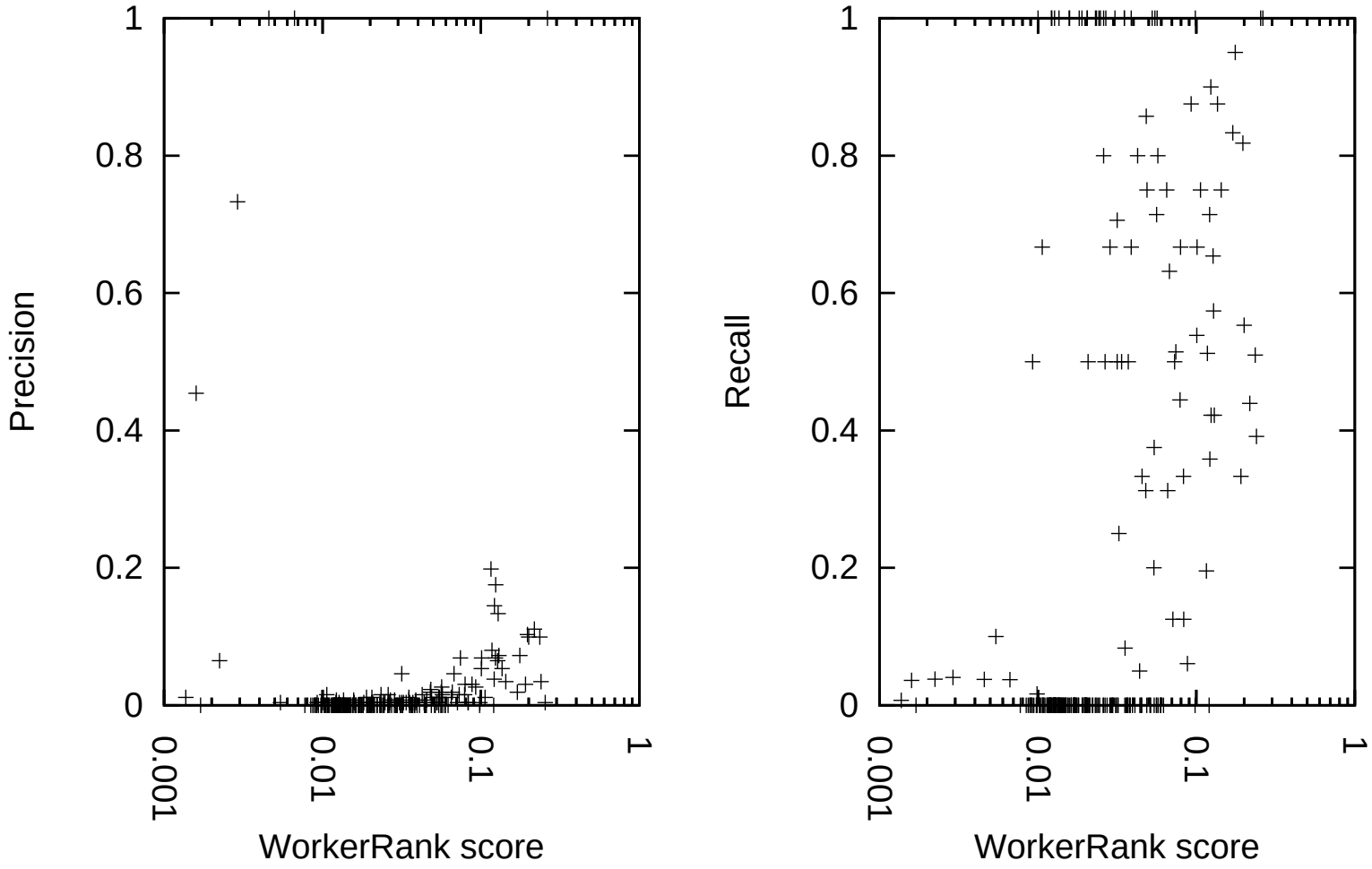

FIG. A.3. Precision and recall for “organization” term, for the positive evaluation method.

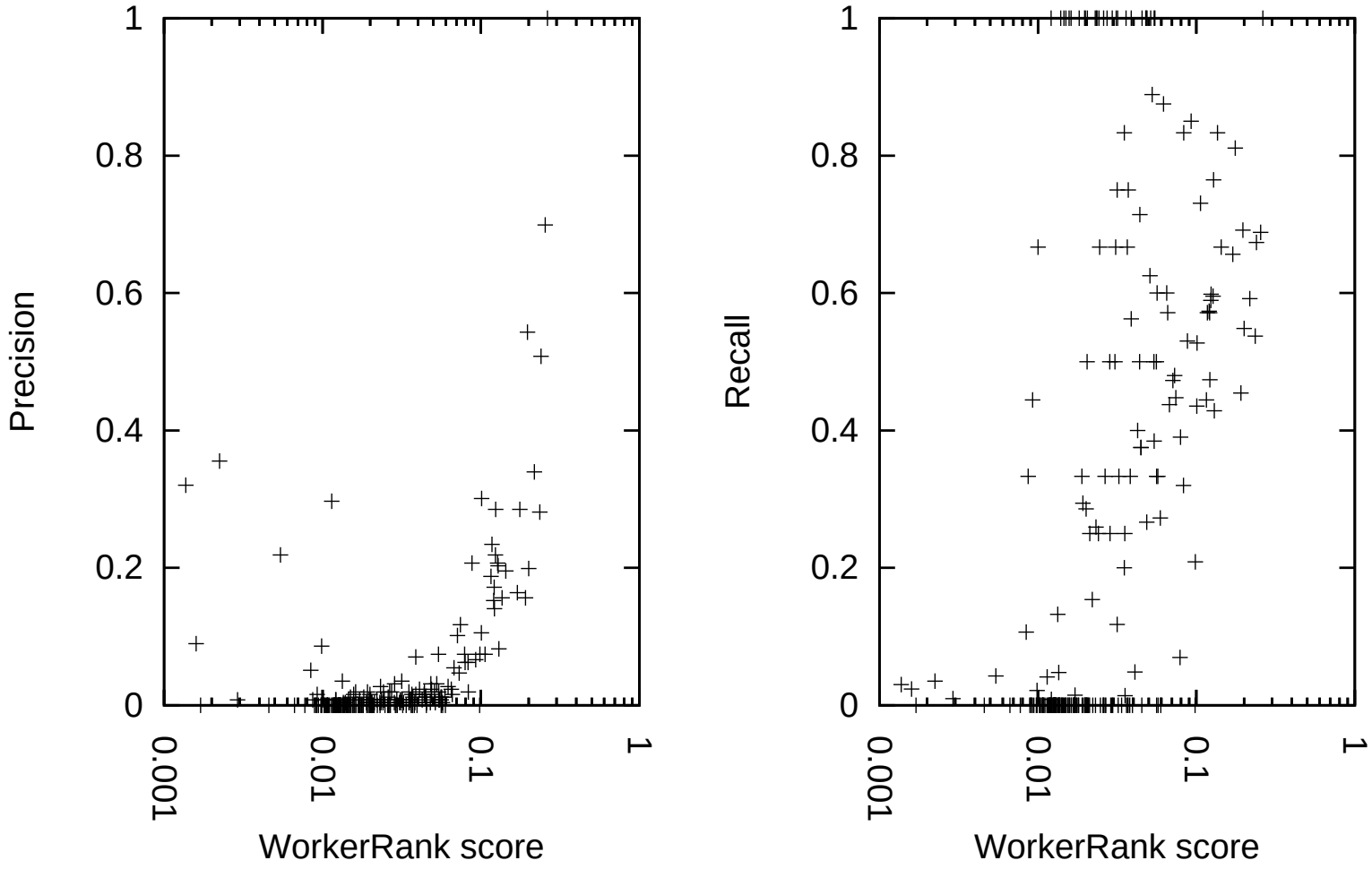

FIG. A.4. Precision and recall for “person” term, for the positive evaluation method.

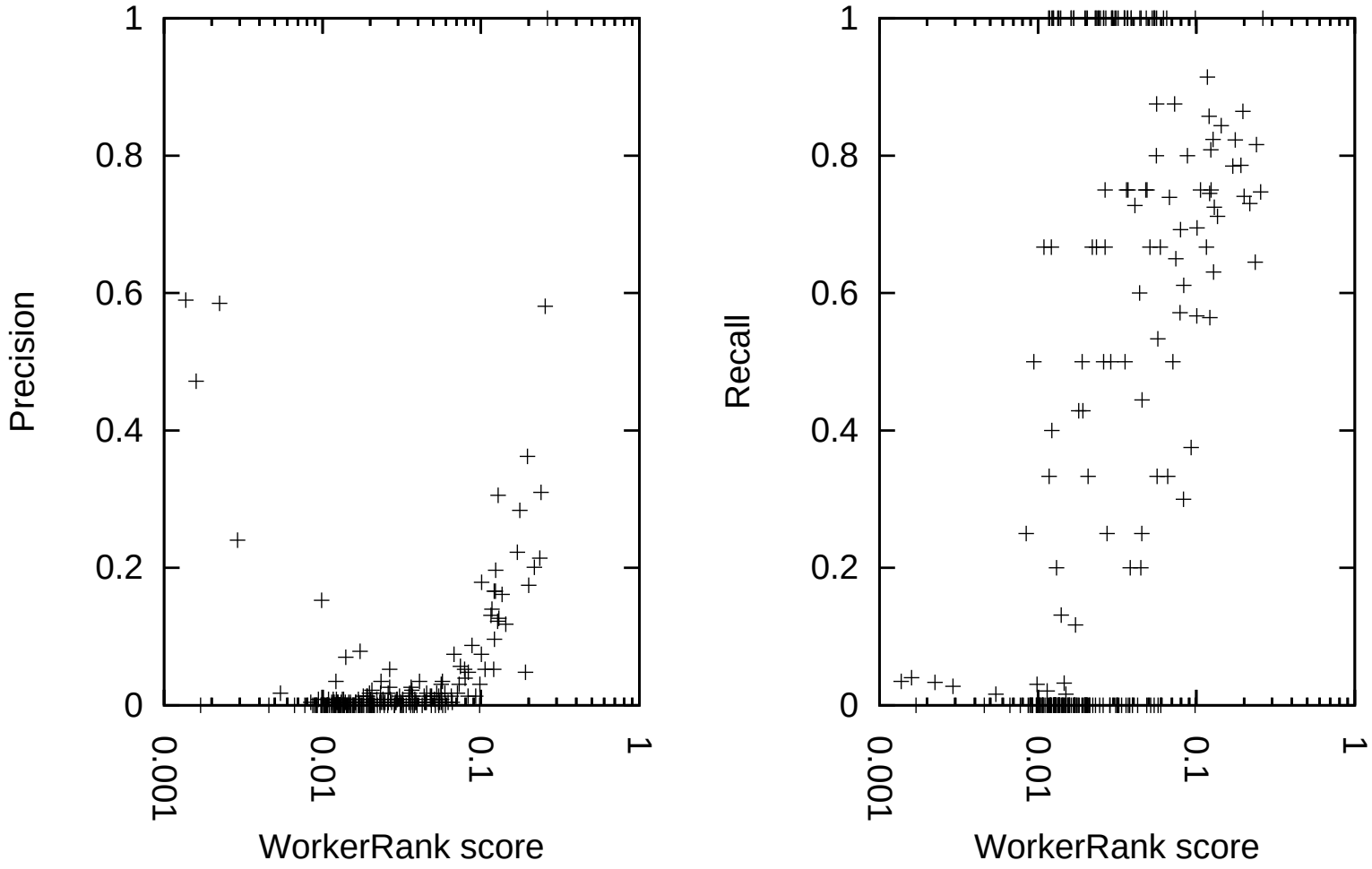

FIG. A.5. Precision and recall for “place” term, for the negative evaluation method.

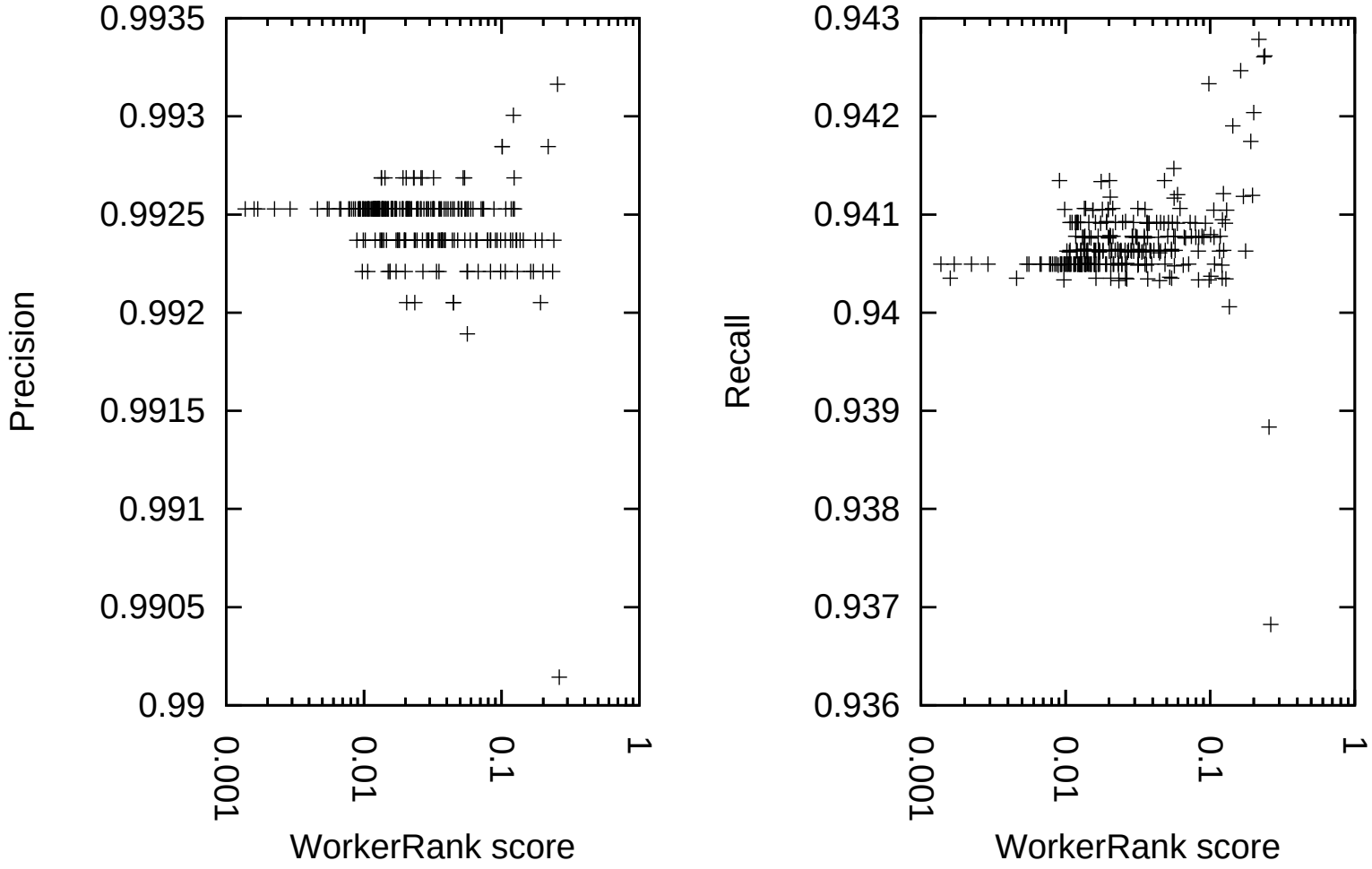

FIG. A.6. Precision and recall for “none” term, for the negative evaluation method.

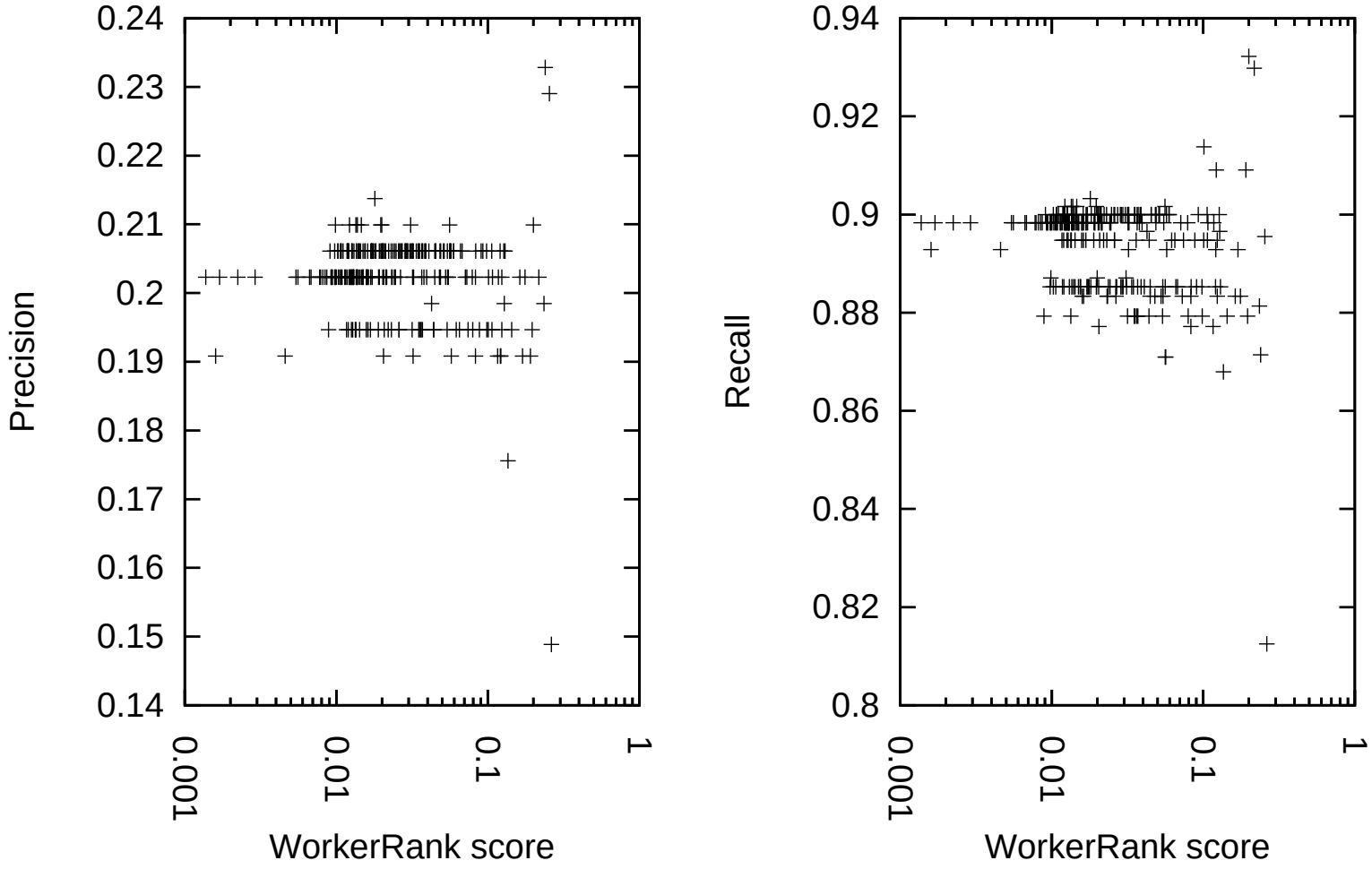

FIG. A.7. Precision and recall for “organization” term, for the negative evaluation method.

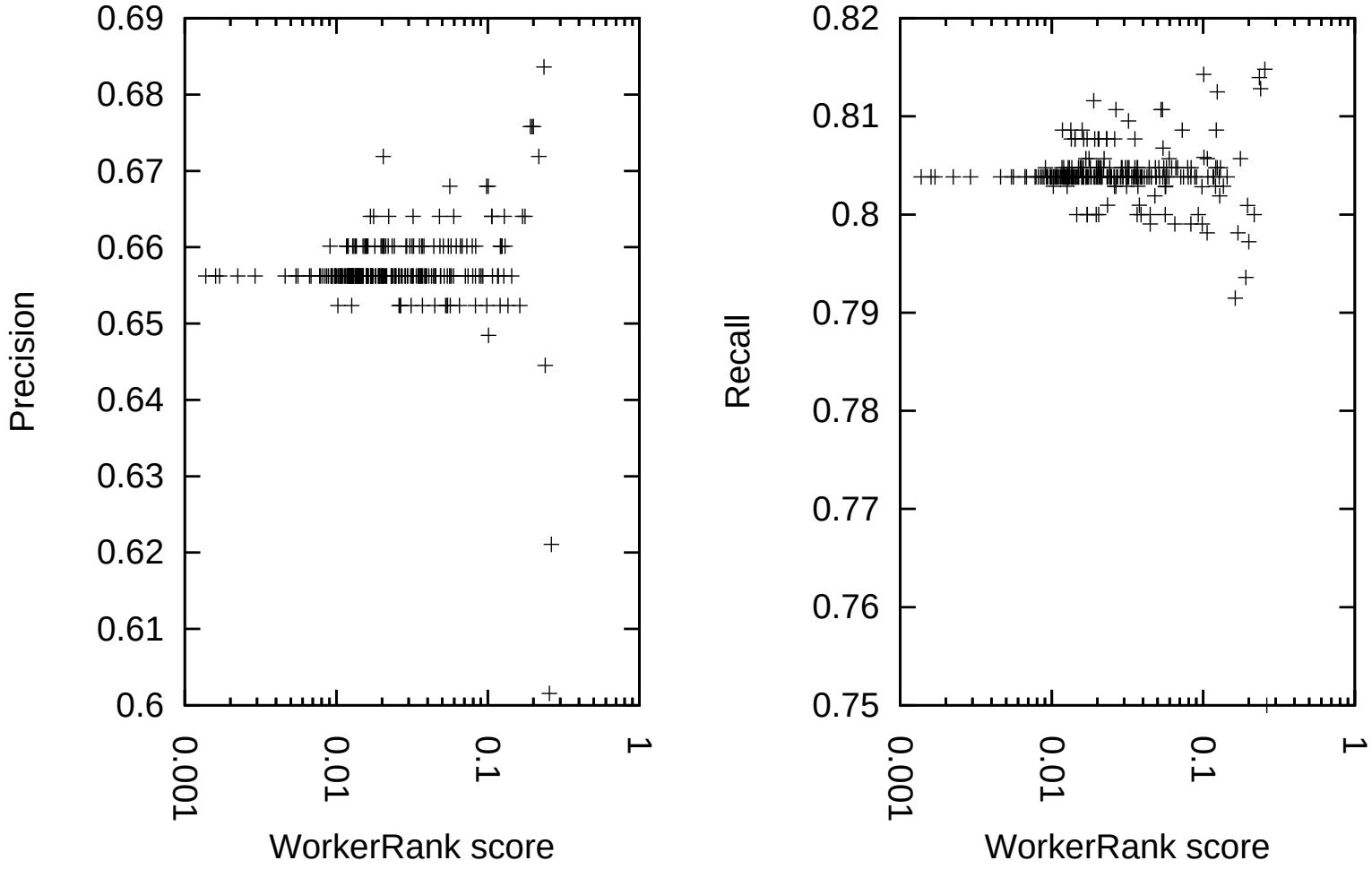

FIG. A.8. Precision and recall for “person” term, for the negative evaluation method.

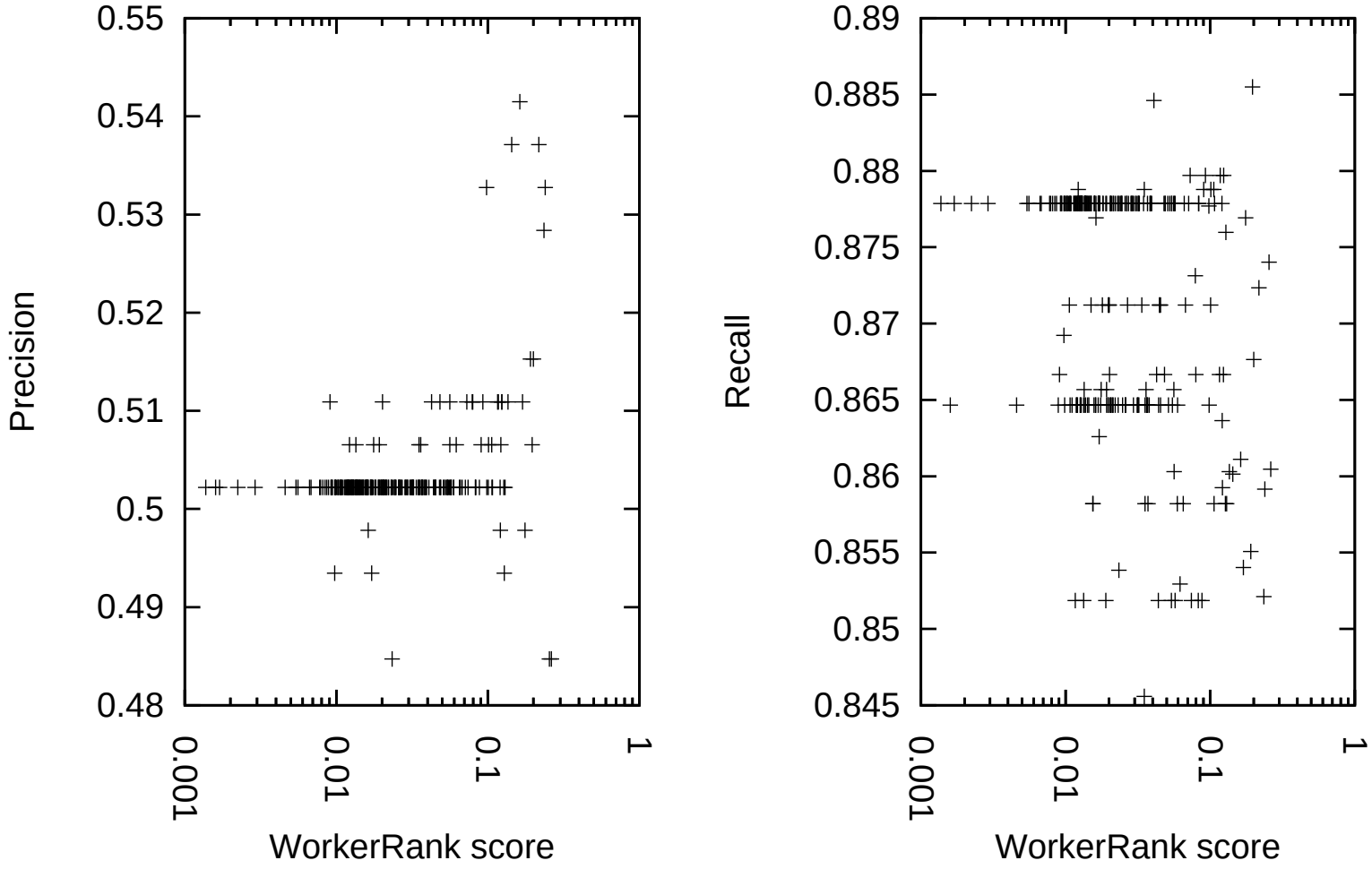

FIG. A.9. Precision and recall for “place” term, for the negative evaluation method.

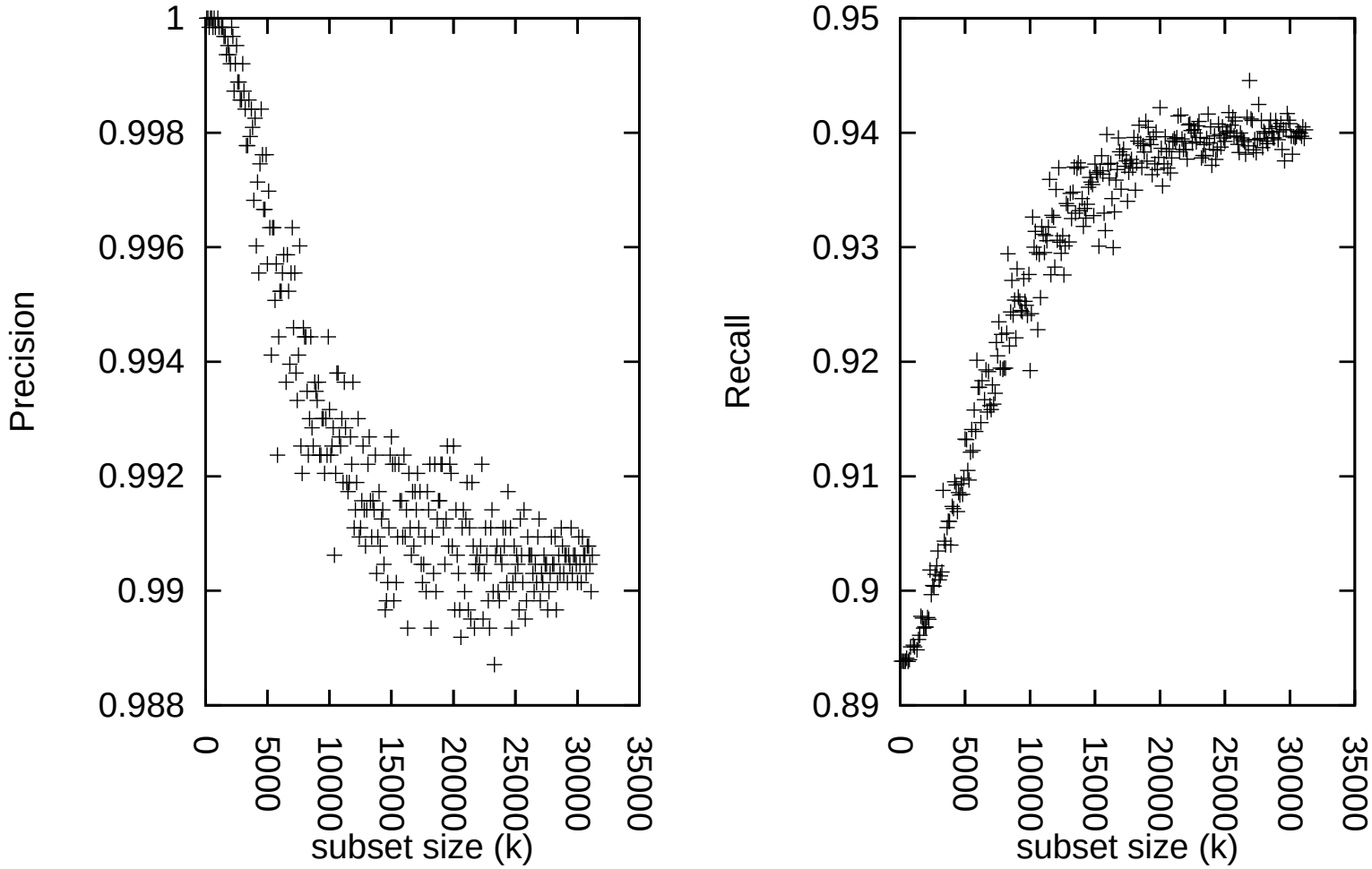

FIG. A.10. Precision and recall for "none" term, on the increasing size-k subsets.

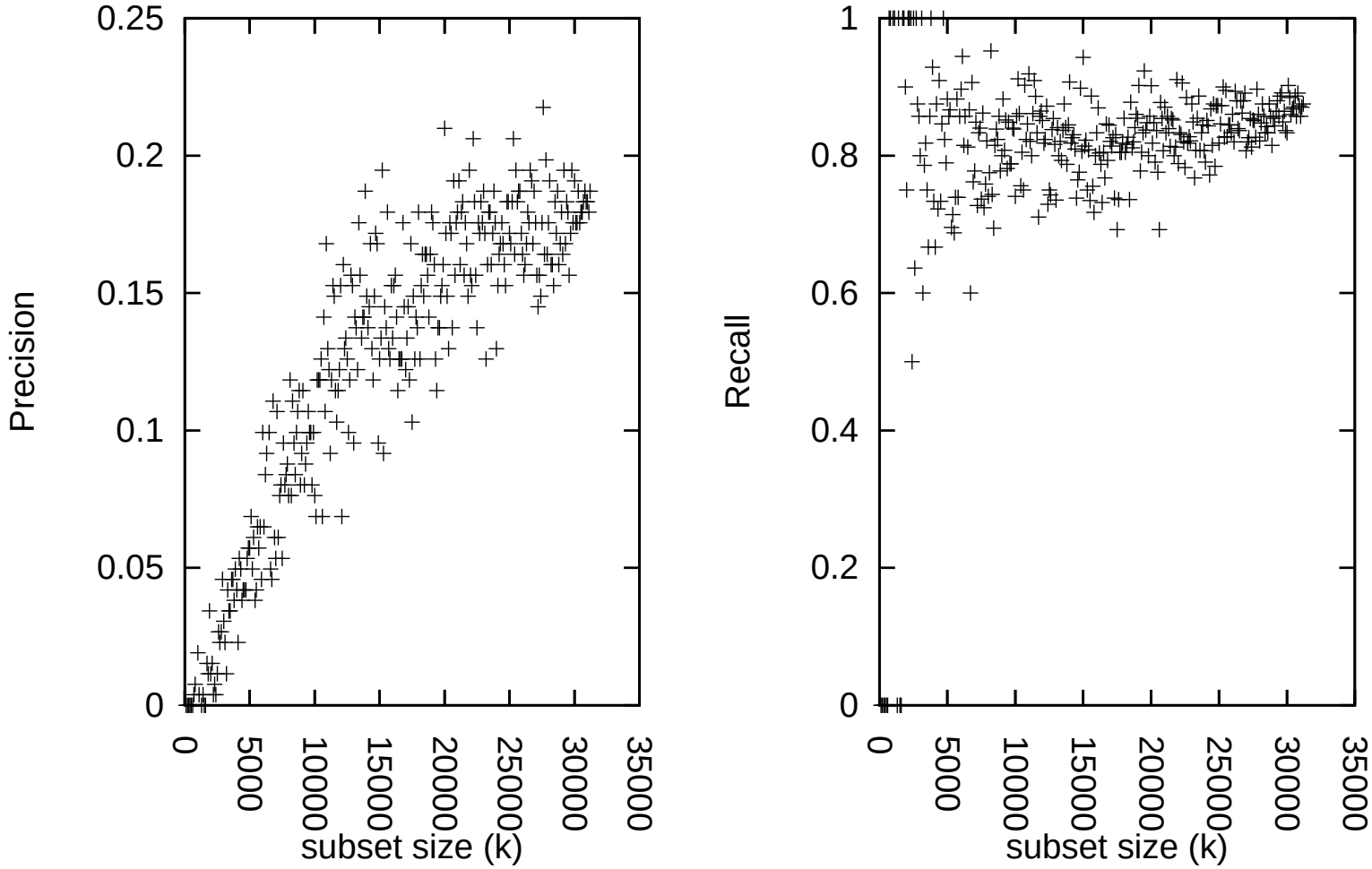

FIG. A.11. Precision and recall for “organization” term, on the increasing size-k subsets.

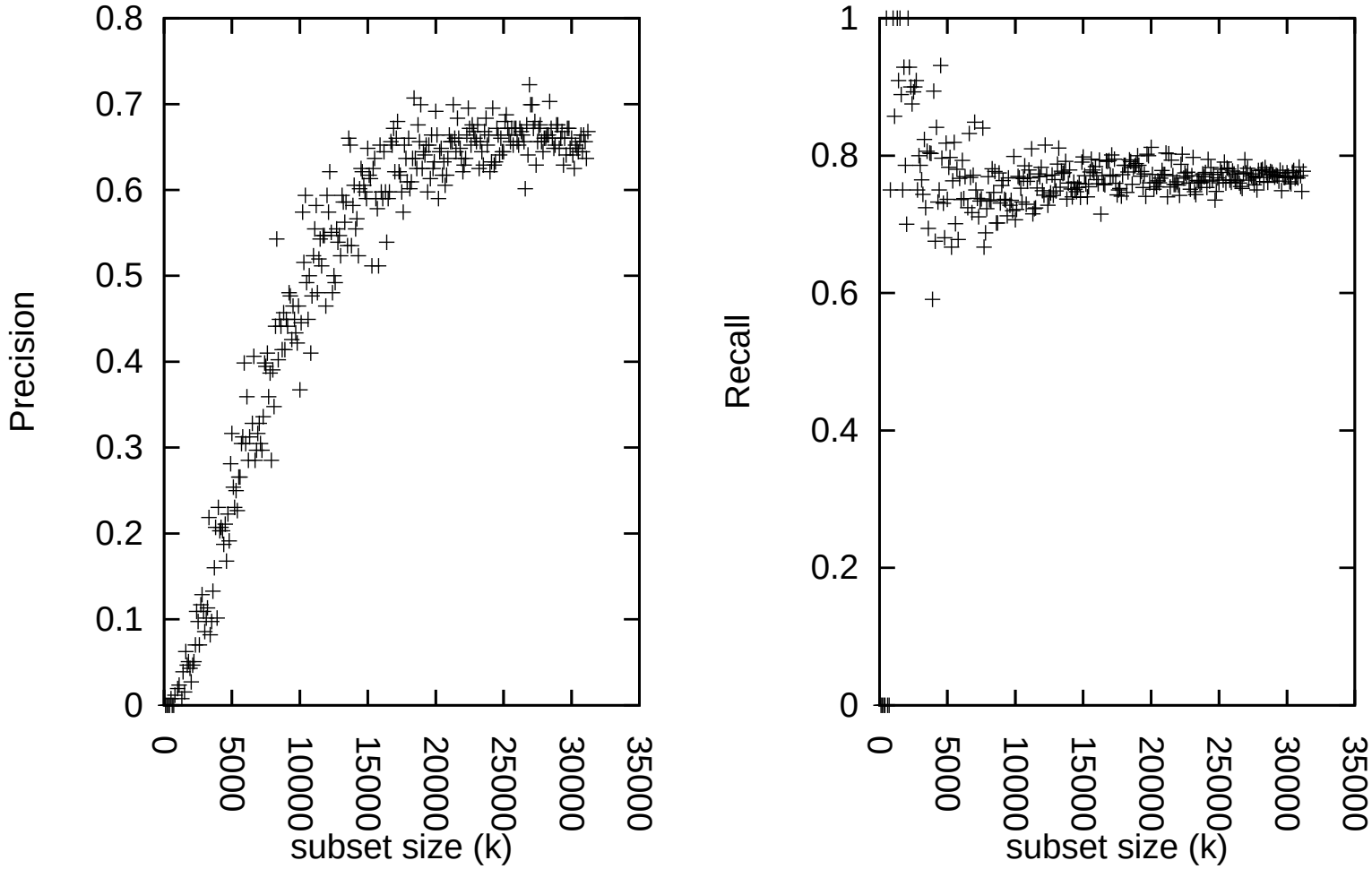

FIG. A.12. Precision and recall for “person” term, on the increasing size-k subsets.

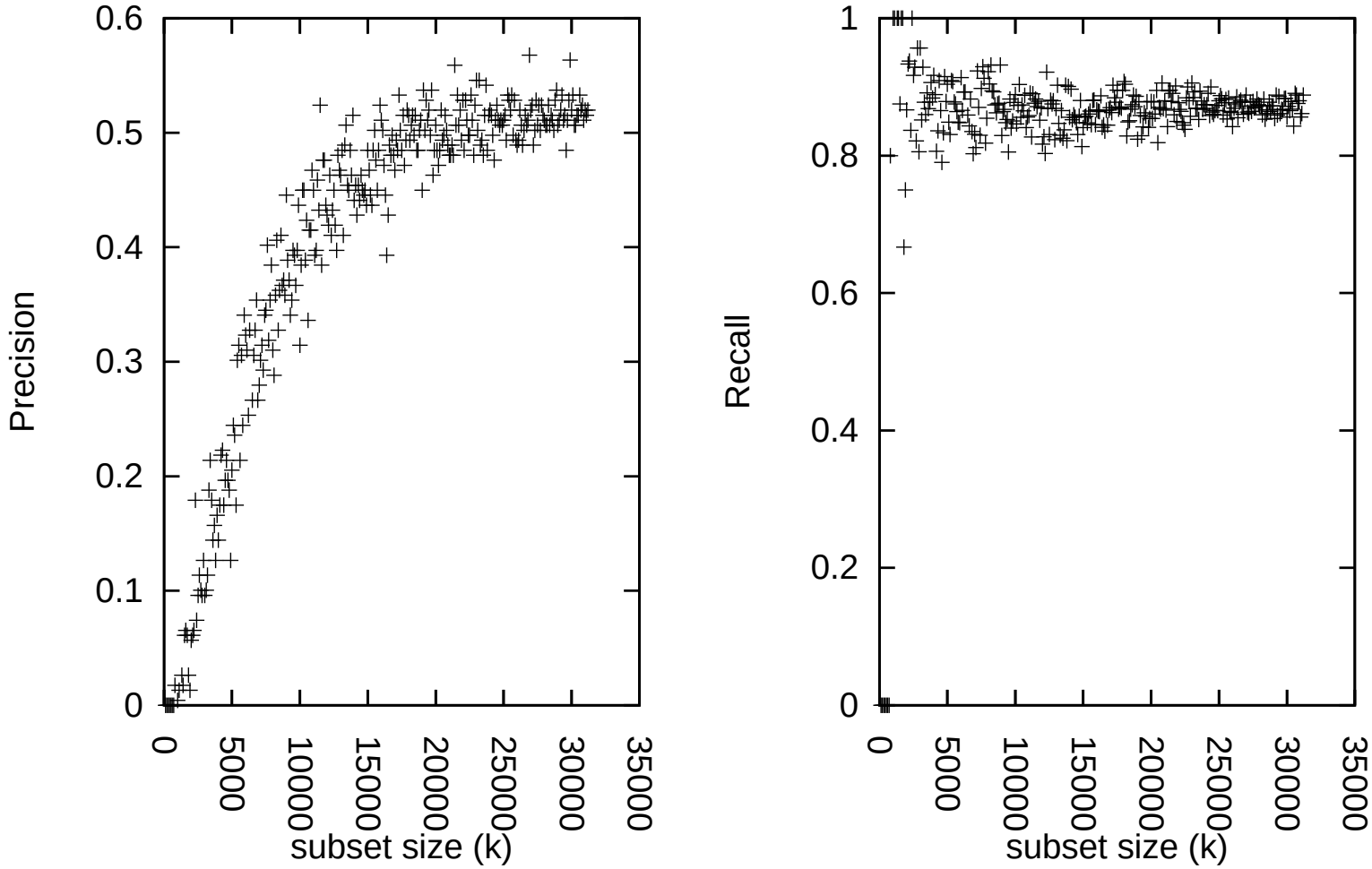

FIG. A.13. Precision and recall for “place” term, on the increasing size-k subsets.

REFERENCES

[1] AAAI. 2008. *BLEWS: Using Blogs to Provide Context for News Articles*.

[2] Amazon.com, Inc. 2010. Amazon mechanical turk faq. `https://www.mturk.com/mturk/help?helpPage=overview`.

[3] Brin, S., and Page, L. 1998. The anatomy of a large-scale hypertextual web search engine. In *Seventh International World-Wide Web Conference (WWW 1998)*.

[4] Cavnar, W. B., and Trenkle, J. M. 1994. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, 161–175.

[5] Facebook.com. 2010. status.set - facebook developers. `http://developers.facebook.com/docs/reference/rest/status.set`.

[6] Finkel, J. R.; Grenager, T.; and Manning, C. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, 363–370.

[7] Friedman, C., and Hripcsak, G. 1999. Natural language processing and its future in medicine. *Academic Medicine* 74(8):890–895.

[8] Group, P. G. D. 2010. Postgresql: The world's most advanced open source database. `http://www.postgresql.org/`.

[9] Jiang, J., and Zhai, C. 2006. Exploiting domain structure for named entity recognition. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, 74–81.

[10] Lafferty, J.; McCallum, A.; and Pereira, F. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

[11] Locke, B., and Martin, J. 2009. Named entity recognition: Adapting to microblogging. `http://www.cs.colorado.edu/department/publications/theses/docs/bs/brian_locke.pdf`.

[12] Mateosian, R. 2009. Twitter. *IEEE Micro* 29(4):87–88.

[13] Pingdom AB. 2010. Twitter: Now more than 1 billion tweets per month. `http://royal.pingdom.com/2010/02/10/twitter-now-more-than-1-billion-tweets-per-month/`.

[14] Snow, R.; O'Connor, B.; Jurafsky, D.; and Ng, A. Y. 2008. Cheap and fast—but is it food?: Evaluating non-expert annotations for natural language tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 254–263.

[15] Sun, Q.; Küçüktunç, O.; Güdükbay, U.; and Özgür Ulusoy. 2007. A natural language-based interface for querying a video database. *IEEE Multimedia* 14:83–89.

[16] Sutton, C., and McCallum, A. 2006. An introduction to conditional random fields for relational learning.

[17] Twitter support. 2010. Twitter support : How to post a twitter update, or tweet. `http://help.twitter.com/forums/10711/entries/15367`.