# Recognizing Meaning in the Crowd: Building Word Sense Inventories on Amazon Mechanical Turk

Master's Thesis

Presented to

The Faculty of the Graduate School of Arts and Sciences

Brandeis University

Department of Computational Linguistics

Anna Rumshisky, Advisor

In Partial Fulfillment

of the Requirements for

Master's Degree

By

Nicholas S. F. Botchan

May 2012

# Abstract

Recognizing Meaning in the Crowd: Building Word Sense Inventories on Amazon Mechanical Turk

A thesis presented to the Department of Computational Linguistics

Graduate School of Arts and Sciences

Brandeis University

Waltham, Massachusetts

Nicholas S. F. Botchan

This thesis explores different strategies for constructing robust, inexpensive and empirically-derived word sense inventories and the corresponding sense-annotated corpus. All strategies explored rely on non-expert linguistic annotations collected through the use of the Amazon Mechanical Turk crowdsourcing marketplace. Experiments using implementation strategies with different quality control mechanisms are reported on in detail. Described herein are multiple best practices discovered through extensive system testing that are required to obtain high quality data given the challenge of using non-expert annotations. Results indicate that

it is possible to obtain sense inventories that correlate with the gold standard, extending it in ways that may prove useful in a variety of other Natural Language Processing tasks.

## Acknowledgements

Special thanks to Anna Rumshisky, James Pustejovsky, Lotus Goldberg, Bert Xue, and Antonella DeLillo for your constant help and support throughout my time at Brandeis.

I would also like to thank my fellow students in the CL program for making it an enormously fun time.

# Contents

CHAPTER 1

# Introduction

Word sense disambiguation (WSD) is an essential component of many natural language processing (NLP) applications that strives to resolve conflicts that may arise from the ambiguity of a single word. However, defining a discrete inventory of word senses can be a difficult task. Historically, this challenge has required expert human annotators often creating ad hoc resources without a clearly defined set of standards, which were often slow and costly. Furthermore, the resulting resources are not always sufficient for addressing the dynamic nature of sense definition or quantitatively capturing how components from several meanings of a word can be activated simultaneously (Hanks, 2000; Pustejovsky, 1995). To address the difficulties described above, a powerful procedure for creating new and task-independent sense inventories is needed to extend and complement existing resources like WordNet (http://wordnet.princeton.edu/).

Described in this thesis are several possible methods for creating robust word sense inventories from scratch that are relatively cheap and fast using the Amazon Mechanical Turk crowdsourcing marketplace (MTurk) (http://aws.amazon.com/mturk/). In recent years, much attention in the NLP community has been given to the subject of using crowdsourcing services such as MTurk to create low cost, high-quality

resources. This thesis is an attempt to extend that research by focusing on the methodological and experimental considerations required to build a lexicon on top of MTurk.

### 1.1. Goals and Hypotheses

The ultimate goal of this research has been to test the feasibility of using MTurk to build a lexical resource that contains precise quantitative information about sense definitions, and can be used across a variety of tasks.

The first hypothesis assumed here is that such a lexical resource is required to advance the state of the art for many NLP tasks. While this research does not directly address this hypothesis, others have shown that, for example, different NLP applications require different sense granularities. Snow et al. (2007) describe a method for merging the fine-grained senses defined in WordNet to obtain the type of coarse-grained distinctions that are typically more useful for tasks like information retrieval (Gonzalo, Verdejo, Chugur, & Cigarrán, 1998) or query expansion (Moldovan & Mihalcea, 2000). As will be described in greater detail in later sections, the data collected for this research has an inherent graph representation, and as a result the granularity of senses can easily be changed by cutting the graph into larger or smaller clusters.

The fundamental hypothesis tested by this research is that it is possible to tap into the linguistic intuition of the non-expert native English speakers available on MTurk to obtain high-quality language annotations at a relatively low cost. To

test this hypothesis, this research attempted to engineer the best possible data collection system, built through an iterative cycle of: implementation ⇒ system testing ⇒ data gathering ⇒ analysis of results ⇒ identifying system improvements ⇒ repeat.

CHAPTER 2

# Related Work

## 2.1. Previous Work

Rumshisky et al. (2009) demonstrate proof of the initial concept of using MTurk to obtain useful WSD annotations. This paper outlines a prototype-based methodology (described in detail in section 5.2) that was manually implemented to collect 516 judgments attempting to disambiguate the verb crush. The resulting data was analyzed in comparison to gold standard data and shown to have high precision and recall (a set matching f-score of 0.93). Preliminaries for creating a complete sense-annotated lexicon using judgments collected on MTurk are discussed at a high level, paving the way for experimentation to determine the optimal system configuration for the task.

Rumshisky (2011) describes in detail the linguistic annotation task that served as the starting point for this research. Several pilot annotations were proposed to determine the optimal number of workers for the task, as well as to determine the utility of various other quality control mechanisms. The desired characteristics of the proposed lexical resource are described, and also presented is the graphical representation that is fundamental to this research (described in this thesis in section 4.1.1).

Early results returned by the first implementation of the data collection system created in the course of this research are presented in Rumshisky et al. (2012). This paper outlines several of the unforeseen challenges that arose from significant changes within the crowdsourcing community in the three years that had elapsed since the original experiment with the verb crush. Lack of initial success and the resulting shift in our experimental strategy are outlined, and preliminary results indicating movement towards higher quality data are presented.

This thesis extends the previous work described above in a few ways. Foremost, the latest results and important new analyses are presented. In addition, this thesis makes explicit system implementation details that have previously been glanced over. Lastly, promising alternative methodologies for data collection are put forward, serving as a launching point for future efforts.

## 2.2. Existing Lexical Resources

### 2.2.1. WordNet

WordNet is a lexical database of English nouns, verbs and adjectives grouped into sets of cognitive synonyms called synsets (e.g. car and automobile form a synset as do dog and canine). Synsets for verbs and nouns are organized hierarchically by hypernym (or *is a*) relationships. A rough trace of the hypernym path through the conceptual network of WordNet for a verb like shatter would look like this: shatter $\rightarrow$ burst $\rightarrow$ break $\rightarrow$ change integrity $\rightarrow$ change. At the top of the hierarchy we

have, most generally, a synset for the concept of change, and every link before it involves some form of change.

WordNet has been used for a variety of different NLP tasks, including word sense disambiguation, information retrieval, text classification, and several others. One feature that makes WordNet useful for such applications is naturally encoded information about similarity between concepts that can be derived from the ontology through path distances. In other words, synsets that are closer together in the ontology are more similar than synsets that are farther apart. For example, the synset for the verb shatter is closer in WordNet to the verb snap than it is to the verb turn–a characterization that agrees with most people's intuition about the meanings of these words.

This type of distance information can be extremely useful, but there is no mechanism currently in WordNet that allows researchers to access the same type of information for senses that exists in the same synset (i.e. how similar are usages of the words dog and canine). In essence, the research laid out in this thesis can be thought of as an attempt to extend resources like WordNet by quantitatively capturing these sorts of relations between different uses of a single word.

### 2.2.2. OntoNotes

OntoNotes is a large, blended corpus containing text from news, weblogs, conversational telephone speech, talk shows, and numerous other sources. It has been annotated with structural information and shallow semantics (word senses linked

to an ontology). Annotation for OntoNotes was done by a team of trained annotators, and a minimum 90% inter-annotator agreement was required for every layer of annotation. OntoNotes was selected as a source of gold standard data for this research because of its wide adoption and stringent annotation guidelines.

### 2.2.3. Corpus Pattern Analysis (CPA)

CPA is a technique of corpus linguistics used to annotate word meanings in text based on phraseological patterns and collocations. It has been used to generate a corpus of sense annotated sentences that serve as a second source of gold standard data for this research.

## 2.3. Crowdsourcing

There has been a tremendous boom in research surrounding crowdsourcing (also referred to as micro-outsourcing), driven in large part by the machine learning community, which often requires high quality data as a basis for building and training more sophisticated software. Three areas of research in the field of crowdsourcing relevant to this thesis are discussed in more detail below.

### 2.3.1. Previous (Linguistic) Data Collection Efforts on MTurk

The following works present a thorough overview of the use of MTurk to collect linguistic annotations:

Snow et al. (2008) describe an early and comprehensive investigation into the use of MTurk as a data collection system. In this work they present an in depth analysis of the data obtained across five linguistic annotation tasks (affect recognition, word similarity, recognizing textual entailment, event temporal ordering, and word sense disambiguation) for which they developed systems to generate HITs on MTurk and process worker results.

Callison-Burch and Dredze (2010) present an overview of the work of multiple researchers participating in a NAACL-2010 workshop which focused on a shared task to create data for speech and language applications with $100. This workshop set out to address several of the open questions surrounding the use of crowdsourcing systems for obtaining linguistic data. These open questions include: What new research and data resources become possible when the cost of creating annotated training data is dramatically reduced? Can untrained, non-expert annotators be used to perform complex annotation? How can researchers obtain high quality annotations at a fair price?

Akkaya et al. (2010) present their findings in an investigation to collect annotations for Subjectivity Word Sense Disambiguation, a task similar to the one described in this thesis that aims to determine which word instances in a corpus are being used with subjective senses and which are being used with objective senses.

Many other researchers have explored the use of MTurk and other crowdsourcing systems in more specific areas of research including general pattern matching

(Tamuz, Liu, Belongie, Shamir, & Kalai, 2011), paraphrasing for machine translation (Denkowski, Al-Haj, & Lavie, 2010), speech transcription (Evanini, Higgins, & Zechner, 2010), evaluation of commonsense knowledge (Gordon, Van Durme, & Schubert, 2010).

Most of this work suggests that it is possible to obtain high quality annotations from crowdsourcing systems with a high agreement to gold standard data. Because workers on crowdsourcing platforms are non-experts, and because the cost per task is generally low, a particular emphasis throughout this body of research is put on implementing the necessary quality control mechanisms to mitigate noisy data introduced by low quality annotators. In addition to this, much of the research also emphasizes the human aspect of crowdsourcing, namely, the importance of designing simple tasks with clear instructions that are easy for people to understand, and the need to communicate frequently and effectively with workers.

### 2.3.2. MTurk Marketplace Analysis

Amazon has failed to make public any information it has about the demographics of workers on MTurk, which often makes it difficult to assess its utility as a data source. This means that none of the traditional information used to screen workers is available to requesters, such as worker age, educational background, skills, interests, etc. Instead, workers on MTurk are identifiable only through a serial number like A2JIU2DEIXBF9R.

To illuminate what motivates workers to complete tasks on MTurk, and to build a profile of the types of people that are available on MTurk, Ipeirotis (2010) posted a survey on MTurk in which he asked 1000 workers to respond to questions about where they were from, what their backgrounds were, and why they completed work on MTurk. In return for their responses, workers were paid $0.10.

The results of this survey indicated that the majority of workers on MTurk are from the US (47%) or India (34%). Requesters trying to design tasks that require fluency in a specific language may want to screen workers, allowing access to only those who come from a country that speaks the language. However, restricting access like this often has the consequence of increasing the amount of time that it takes to get complete data.

### 2.3.3. Worker Quality Control

Determining worker quality is of critical importance to all but the most simple MTurk applications. Having a good measurement of how workers are performing at a given task allows requesters to reward the best workers, screen poor quality workers, and modify the system as needed.

Unfortunately, determining the quality of submitted results can sometimes be as difficult as the original task itself, a fact that malicious workers may try to take advantage of by submitting answers of low quality. These bad workers are commonly referred to as spammers.

Early systems relied on redundancy to verify correct answers. Snow et al. (2008) showed that high agreement between expert and non-expert judgments on certain linguistic tasks emerged when data from 10 different Turkers was collected for each individual label. They also showed that agreement could be increased further by weighting worker votes based on their performance compared to a gold standard (or the majority vote). However, this sort of redundancy (10 annotators) can be expensive, and is not always an option. Thus, sophisticated methods for determining worker quality are needed.

In general, to determine worker quality it is not enough to analyze only the accuracy of a worker against gold standard data. For example, if 80% of the data in a categorization task falls into one category, then a worker who places everything into that category automatically has 80% accuracy, despite the fact that they are not providing any meaningful input. The WSD task described in this thesis exhibits this sort of skewed distribution. The words we are interested in have a high degree of polysemy, and therefore it is most likely that when comparing two different examples of a word in context, their meanings will come from different senses.

What is needed therefore is an estimate of true worker error rate, or in other words, a measurement of how much it costs a worker to switch from an incorrect label to a correct label. Ipeirotis et al. (2010) describe in detail an algorithm for separating out true error rate from the noisy biases that some workers exhibit. The main procedure to estimate worker cost runs as follows: for each worker

prior probabilities of assigning a label i to an object are computed. Next, for each assignment, the best possible estimate for the true label of the assignment is computed (when gold standard data exists, the gold standard label is used– otherwise a weighted majority vote is used). Given the true label, the expected cost of a worker assigning that label is calculated. Finally, knowing how often the worker assigns a label and the expected cost, the average misclassification cost of each worker is calculated, a scalar value for which perfect workers have costs of zero and random workers or spammers have high expected costs.

CHAPTER 3

# Resources

## 3.1. Amazon Mechanical Turk

Amazon Mechanical Turk (MTurk), a branch of Amazon Web Services, is an internet crowdsourcing marketplace where computer programmers (called requesters) can create and post tasks for workers to complete piecemeal for money. It has been branded as a platform that enables requesters can tap into human intelligence to collect data or perform tasks that computers are unable to do currently. These tasks are referred to as Human Intelligence Tasks (HITs), and the most popular examples used to describe the sort of HITs that MTurk is designed for include asking workers to identify which pictures of storefronts look the best, or to classify pictures of objects into their appropriate categories (cars, birds, etc.).

The basic usage of MTurk is as follows: requesters create a number of HITs that are publically available to registered MTurk workers (sometimes called Turkers). Workers find these HITs typically by performing a keyword search. If a worker is interested in a task based on its description and a preview of the work, they may choose to accept the HIT, in which case they have a pre-specified amount of time to complete the work. When a worker finishes a task, results are sent back to the

requester who then must approve or reject the assignment, resulting in payment or withholding of payment to the worker.

### 3.2. Boto (.mturk)

Boto is an open source third-party Python interface to Amazon Web Services. It contains an mturk module, which implements the operations and data structures described by the official MTurk application programming interface (API). This enables Python programmers to design software for handling the entire data collection and validation process associated with MTurk. The architecture for this research relies heavily on the boto.mturk module.

### 3.3. GetAnotherLabel

As described in section 2.3.3 above, the results returned from resources like MTurk are of imperfect quality. Workers may make a variety of different kinds of mistakes; some workers may in fact be trying to take advantage of the system by doing minimal, if any, real work (spammers).

GetAnotherLabel is a free tool available online that allows requesters to infer the underlying quality of workers, for a given task, by statistically examining the labels submitted by the workers. It enables requesters to examine which workers are doing a good work for a given task, and in turn reward these workers appropriately. The tool also provides the ability to detect spammers, allowing the requester to take action to mitigate the noise produced by low quality workers. The idea

behind this software is outlined in the Worker Quality Control section above, and is described in detail in Ipeirotis et al. (2010).

### 3.4. MCL

MCL is a free implementation of the widely adopted Markov Clustering Algorithm (mcl), which is used to produce graph clusters in a general setting. This unsupervised clustering algorithm cuts a graph based on simulation of (stochastic) flow in a graph across weighted edges. The tightness and amount of clusters generated as output from the algorithm can be easily manipulated by varying the inflation parameter that it takes as input. Typically, higher inflation values result in fine-grained clusterings, and lower inflation values result in coarse-grained clusters.

### 3.5. SQLite3

SQLite3 is a lightweight relational database management system that understands most of the standard SQL language. It has several attributes that made it ideal for this research. It comes prepackaged with most versions of Python, providing an SQL interface that can easily be integrated into Python scripts and applications. In addition to this, SQLite does not have a separate server process. Instead it reads and writes directly to ordinary disk files, meaning that an SQLite database with multiple tables, views and indices gets stored in a single file that can be freely copied across platforms. As such, it is optimized for rapid prototyping

and testing systems under uncertain conditions. The majority of results for this research are stored in SQLite database files.

## 3.6. Pydot and GraphViz

Pydot is a Python interface to GraphViz, an open source graph visualization toolkit. The graph visualizations presented here were generated through pydot.

## 3.7. Evaluation Metrics

### 3.7.1. F-measure

The F-score (Zhao, Karypis, & Fayyad, 2005; Agirre & Soroa, 2007) is a set-matching measure used in this thesis to compare the senses that get created via data collected from MTurk to gold standard senses. Precision, recall, and their harmonic mean (van Rijsbergens F-measure) are computed for each experimentally generated sense/gold standard sense pair. Each generated sense is then matched to the gold standard class with which it achieves the highest F-measure. The F-score is computed as a weighted average of the F-measure values obtained for each generated sense (weighted by the size of the generated sense).

### 3.7.2. Fleiss' Kappa

Fleiss' kappa (referred to hereafter as just kappa) is an inter-annotator agreement measure. It is formally defined as follows (from (Fleiss et al., 1971)):

$$\kappa = \frac{P_o - P_e}{1 - P_e}. \tag{3.1}$$

In equation 3.1, the denominator $1 - P_e$ gives the degree of agreement that is attainable above chance, and the numerator $P - P_e$ gives the degree of agreement actually achieved above chance. The kappa ratio can therefore be thought of as a measurement of how reliably annotators are in agreement when compared to a baseline level of chance agreement.

More formally, let $N$ be the number of documents or instances to be annotated, let $n$ be the number of ratings or annotations obtained per document, and let $k$ be the number of categories into which assignments are made. To compute Fleiss' kappa, an $N$ by $k$ observation table is generated. In this table, each of the $i$-th rows represent a document, each of the $j$-th columns represent a category, and each of the elements $n_{ij}$ in the table represent the number of observers who assigned the

$i$-th document into the $j$-th category. Definitions for $P_e$ and $P_o$ are as follows:

$$p_j = \frac{1}{Nn} \sum_{i=1}^{N} n_{ij}, \tag{3.2}$$

$$P_i = \frac{1}{n(n-1)} \sum_{j=1}^{k} n_{ij}(n_{ij}-1) = \frac{1}{n(n-1)} \left( \sum_{j=1}^{k} n_{ij}^2 - n \right), \tag{3.3}$$

$$P_o = \frac{1}{N} \sum_{i=1}^{N} P_i, \tag{3.4}$$

$$P_e = \sum_{j=1}^{k} p_j^2, \tag{3.5}$$

where $p_j$ is the proportion of all assignments to the $j$-th category, $P_i$ is the extent of agreement among the $n$ annotators for the $i$-th document, $P_o$ is the observed overall agreement, and $P_e$ is the expected mean proportion of agreement due to chance.

CHAPTER 4

# System Architecture

This section outlines the most current version of the system for collecting and analyzing results. It is instructive to see how the system has evolved over the course of this research, so included in this chapter is an additional section that describes previous iterations of the system with some explanation as to why changes were made.

## 4.1. Task Specification

The basic task we ask workers on MTurk to perform is the following: Workers are presented with a single sentence that uses the word to be disambiguated (highlighted in red). They are then asked to compare the usage of that word to several other example sentences using the same highlighted word. For each comparison, workers must determine if the meaning of the highlighted word is the same, different, or if it is unclear between the two sentences. Figure 4.1 below shows an example of such a HIT.

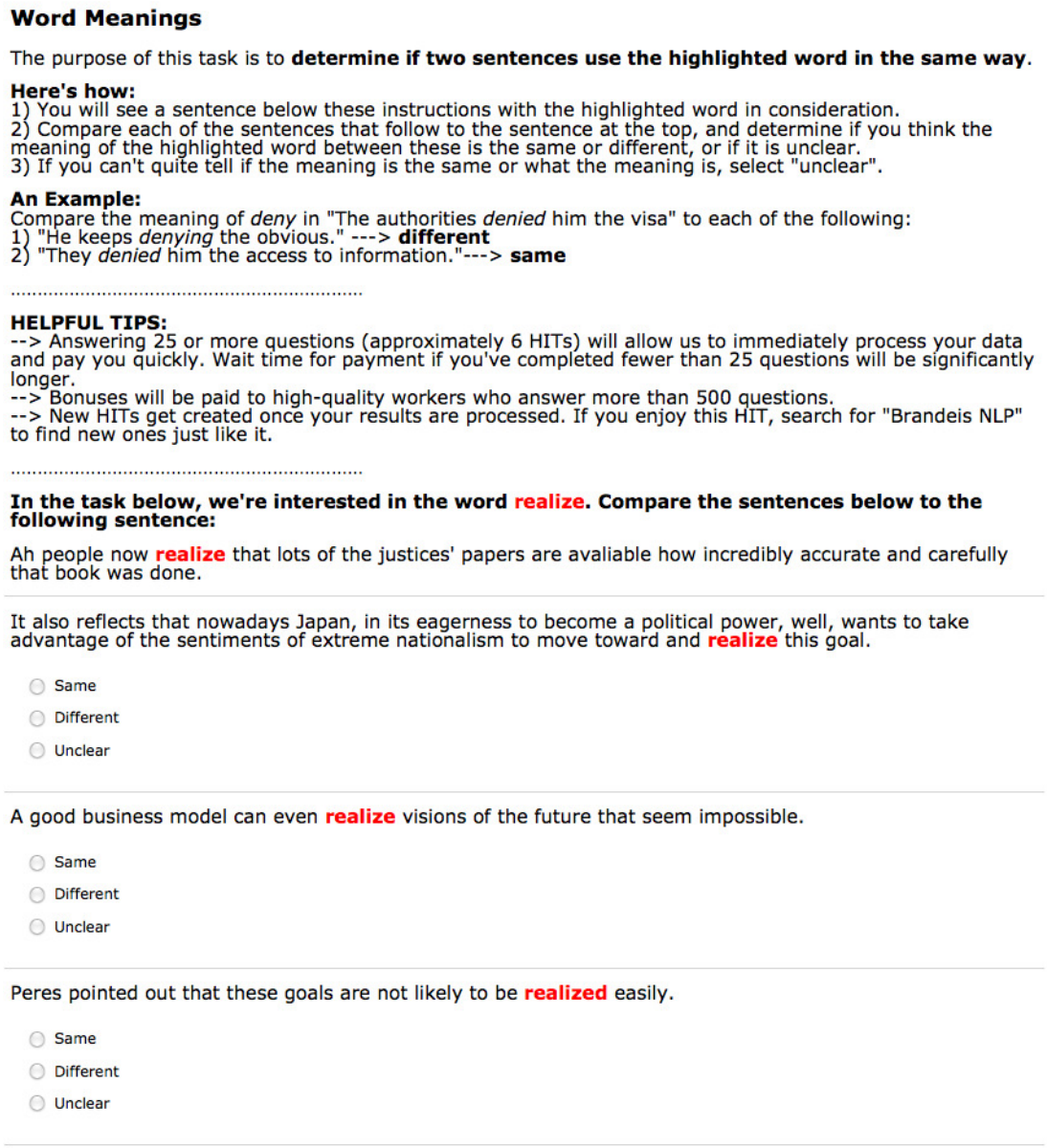


Figure 4.1. A sample Human Annotation Task (HIT) generated for this research.

### 4.1.1. Graphical Representation

The data collected for a single word has an inherent graph representation in which nodes are sentences, edges are similarity judgments (weighted by how many annotators judge the two sentences they connect as similar), and interconnected clusters of nodes correspond to different senses for the word. In this representation, every word has a different graph. Figure 4.2 show two such graphs for the verbs rain and shine.

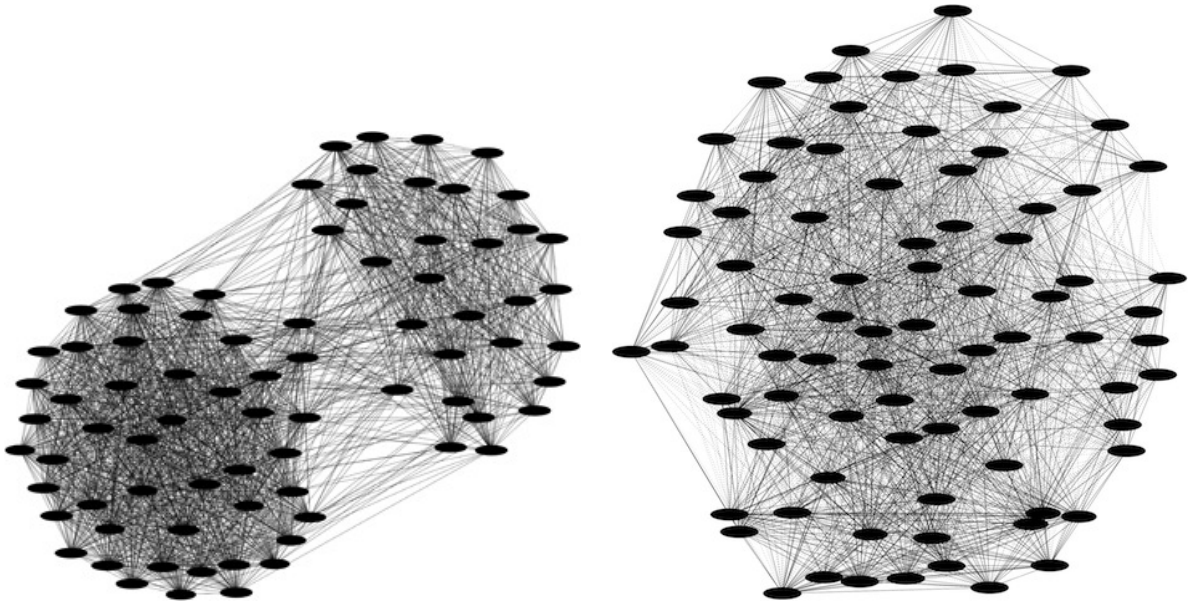

Figure 4.2. Graphical representations for the verbs rain (left) and shine (right). Finer-grained clusters can be discerned from these highly interconnected graphs.

### 4.1.2. Terminology

Throughout the rest of this thesis, the term experiment is used to refer to all of the data associated with the crowdsourcing of judgments for a single lexical item, as the task is described above. An experiment is defined by a word, a part of speech, and a set of sentences that use the word and need to be to be compared to each other on MTurk. The judgments collected through MTurk are referred to as the experimental results or data. These terms are derived from the fact that the judgments collected for a word can be compared to a gold standard, and thus the process of posting HITs and analyzing the results is a controlled experiment.

Following the convention that Amazon uses, we make a distinction between HITs and assignments. In the MTurk API, HITs specify the questions to be asked to workers on MTurk, and assignments are worker responses to HITs. A HIT can specify the number of workers that should (redundantly) complete the task, and for each worker that completes that HIT a different assignment is returned by MTurk.

## 4.2. System Overview

The system used to collect experimental results from MTurk was developed entirely in Python 2.6.5, and relies on multiple third party open source toolkits mentioned in the resources section above. For programmatic access to the MTurk API, the system relies on the Python boto.mturk module.

There are four main pieces to the system, each piece corresponds to a different stage in the data collection process. Although it makes sense to describe the system in terms of these four pieces, in reality, the system is spread across 26 separate modules consisting of several thousands of lines of code.

The four main pieces of the system are organized as follows: The pre-processing stage consists of picking verbs to test, and making sure that data is properly formatted for MTurk. Next, experiments are initialized and HITs are programmatically generated and posted to MTurk for workers to complete. The most critical stage follows–worker results are processed, and these results are analyzed for quality control purposes. Lastly, after all data has been collected, results are evaluated and new models for development and data collection are prepared for testing.

Each piece of the system is highly configurable, and details about the optimal configuration discovered throughout this research are discussed throughout the remainder of this chapter.

## 4.3. Pre-processing

### 4.3.1. Generating Verb Candidates

Before HITs can be posted to MTurk, a list of candidate verbs is generated based on the following criteria:

(1) the number of senses for the word in the gold standard

(2) the distribution of example sentences across senses

(3) the total number of example sentences available

(4) the number of example sentences in the smallest sense

The goal of using these selection criteria is to find a representative subset of verbs that have a high degree of polysemy and a wide variety of rather common usages. Light verbs with very many senses, verbs that do not have enough example sentences, and verbs that have only one major usage have been discarded from experimentation based on this selection criteria.

### 4.3.2. Downsampling

Several of the earlier experiments conducted for this research were based on verbs that had a very large number of example sentences distributed across multiple senses. This was problematic chiefly because obtaining annotations between several hundred example sentences is prohibitively expensive (collecting a full pairwise set of comparisons between 150 example sentences would cost approximately $160). To mitigate this problem, a solution was implemented in which the total number of sentences to be compared on MTurk was reduced to a specified target that could be adjusted based on how much money was available for the data collection effort.

A rough outline of the downsampling function goes as follows:

**Input:** A large list of sense annotated (gold standard) example sentences for a given word; a target number of sentences (typically between 80 - 150).

**Output:** A list containing the target number of example sentences in which the original distribution of example sentences across senses is approximately preserved.

**The downsampling process:**

**Initialize Bins:**

Initialize a bin of example sentences for each sense by randomly selecting from all sentences (without replacement) a minimum number of example sentences for each sense (e.g. 10 sentences per sense).

**Compute Sense Distribution:**

For each sense, calculate what fraction of all example sentences use that particular sense.

**Fill Bins:**

With the remaining space (target minus the total number of sentences placed in initialized bins), fill up each bin so that the fractions calculated above still approximately hold.

### 4.3.3. Preparing Data for MTurk

After downsampling, sentences need to be cleaned and formatted in HTML so that they appear correctly on MTurk. This step involves quoting special characters (e.g. the & character, which has special meaning in HTML), and adding HTML emphasis tags around the ambiguous word in consideration to increase the visual clarity of the task.

To properly interpret requesters HITs, Amazon requires that HITs be specified in an XML formatted data structure called a question form. Question forms consist of two main sections: an overview section (with the HIT title, HIT keywords, and

a general description of the task), and a list of XML formatted questions to be answered by workers.

Question forms for HITs can be built programmatically through the Python boto.mturk package, which comes with a QuestionForm class and other useful classes for this task. After initializing a new question form object, an overview object (i.e. a task description) gets appended to the question form along with a list of individual question objects. Each individual question object specifies the answer style (radio-button), and is flagged as mandatory for completion. To process results on the back end, each question also needs to be annotated with a unique identifier. Workers do not see this unique identifier, making it useful place to upload information to MTurk that can be used for reviewing the HITs. The unique identifier of a question is of the following format:

**lemma–part of speech–source–sentence 1 ID–sentence 2 ID**

A few other things need to be specified before HITs can be posted to MTurk. Requesters have the option of using qualification requirements to ensure that workers are qualified to complete the HITs being created. Amazon provides several useful built in qualifications such as locale, percent of assignments approved and number of HITs approved, and it is considered good practice to use these qualification requirements as a preliminary quality control mechanism. In addition to these built in qualification requirements, requesters can create their own qualification requirements that test workers abilities for a given task.

The default qualification requirements for HITs that get created by the system for this research are as follows: locale is restricted to the US only, workers must have a previous hit approval rate of 85%, and workers must have a total of over 200 HITs previously approved.

Lastly, and perhaps most importantly, requesters must decide on how much to pay workers per HIT. The art of determining exactly how much to pay workers depends on several factors, namely, how much money a requester has at their disposal, how long they are willing to wait to get data, and how stringent they need the results to be. After a good deal of experimentation, a price of $0.03 per HIT (10 comparisons) was settled upon for this system.

Because it is difficult to determine what the optimal HIT design will be for any given task, the part of the system that regulates HIT design and price is easily configurable. Later sections describe in more detail configuration options and experimental results that led to the current default values for these parameters.

## 4.4. Data Collection

### 4.4.1. Main Processing Loop

The following is a rough sketch of the main processing loop used to collect results from MTurk.

While experiments are incomplete:

**Scan for Results:**

Scan over all posted HITs and add any completed assignments to the database

**Update Worker Statistics:**

Update worker statistics based on any newly added assignments

**Clean Out Bad Data:**

Remove bad judgments (submitted by low quality workers) from the database

**Administer Payments:**

Approve, reject, and extend assignments (or create new HITs) as needed, pay bonuses to the best workers

It is straightforward to determine when a full pairwise experiment has been completed because every comparison that needs to be made is known beforehand. For other possible experiment types, determining when an experiment is complete may be less straightforward because HITs may need to be generated dynamically based on the experimental strategy.

### 4.4.2. Worker Quality Assessment

Error rates for workers are calculated using the java based GetAnotherLabel tool designed by Panos Ipeirotis. This software takes as input the following data:

(1) An input file containing labels assigned by workers

(2) A correct file containing the correct (gold standard) labels for some of the objects in the data

(3) A cost file containing misclassification costs (i.e. the penalty of incorrectly classifying an object of category A as category B)

In our system, the misclassification costs for all possible misclassifications is set to one, except when something is misclassified as unclear, which has a penalty of 0.5. Setting the unclear misclassification cost as lower than all other misclassification costs is a necessary compromise due to the fact that the gold standard data is broken into hard clusters with no information about when a word usage is between senses. Thus, legitimate workers may be correct in saying that the difference in the usage of a word between two sentences is unclear, but this judgment compared to the gold standard will always be wrong. While this is not an ideal set up, it is necessary to reduce this type of misclassification cost to prevent over penalizing legitimate unclear judgments, and it should be pointed out that there is a strong bias against using the unclear judgment exhibited by all workers.

Worker error rates cannot be accurately estimated without sufficient data. The system therefore does not make any error rate estimations for a worker until at least 25 judgments have been collected from that worker. This can be slightly problematic because workers typically expect their work to be approved promptly, and if a worker completes fewer than 25 judgments the system will generate quality estimate for that worker, and therefore their work will not be automatically approved or rejected.

### 4.4.3. Approving/Rejecting Assignments

Due to the high volume of data returned by MTurk, it is necessary to automatically approve or reject worker assignments in order to pay workers promptly and fairly. A single full pairwise experiment with 80 example sentences and 5 annotators has a total of 15,800 worker judgments that need to be reviewed and compared to the gold standard.

Assignments are approved or rejected by the system automatically depending on the worker error rate estimates discussed above. After enough annotations from a worker have been collected (for the error rate estimates to be statistically significant), if a worker has a low error rate, then all of their work up to that point is approved and they are paid for their work. Conversely, if a worker's error rate is too high at the outset, then all of their work is rejected, their poor judgments are removed from the database, and the corresponding HITs are extended to allow for another worker to attempt to do a better job.

Occasionally a worker with an initially low error rate may start to perform poorly. If a previously good worker has an error rate that moves above the acceptable threshold, all previous work from that worker is kept, but incoming work from that worker is rejected.

### 4.4.4. Blocking Low Quality Workers

Rather than using the BlockWorker function defined in the MTurk API, which can result in deactivation of a worker account and completely blocks workers from performing any of the tasks submitted under the umbrella BrandeisNLP requester account, a slightly less harsh mechanism is used to prevent low quality workers from accessing HITs for this research.

Each experiment has an automatically generated qualification requirement that is used to track worker error rate. At the outset of an experiment, all workers automatically have a perfect score for this qualification requirement, granting them access to the HITs. Workers who have a score for this qualification requirement below a set threshold value are no longer qualified to work on the associated HITs, and therefore cannot access them. After initialization, as workers complete annotations, the qualification score for each worker is updated automatically based on the error rate estimates discussed above. This allows the system to prevent spammers and bots from accessing HITs for this research without over penalizing legitimate workers who just arent good at the basic task.

## 4.5. Redundancy and scaling considerations

This system was designed as a prototype for a large scale annotation effort. It has several redundancies designed to make sure data gets processed correctly and to prevent issues that may arise from service interruptions. In general, these redundancies lead to an increase in processing time that may not be desirable for

a large scale effort. However, the true limiting factor for any mturk system is how long it takes workers to discover and complete HITs generated by the system, which is typically several orders of magnitude larger than system processing time.

For this thesis, the system never had more than 1500 outstanding HITs posted to MTurk. To process completed assignments, the system scans over each outstanding HIT and pulls all completed assignments for that HIT. One of the redundancies is that at every pass over outstanding HITs, all completed assignment for a HIT are tested against the database. This is intended to prevent issues that can arise from system interruptions, resulting in things like partially filled database tables and other inconsistencies. However, it means that the same completed assignments get tested repeatedly in an inefficient manner. With 1500 HITs posted to MTurk, this inefficiency was never an issue. However, at a much larger scale a more sophisticated data integrity testing mechanism would be needed to prevent system slowdowns that could lead to untimely processing of worker assignments.

The database structure of the system was developed to maintain modularity between different components of the system, and as a result is not normalized. For example, uncertainty in how to approach testing worker quality led to the development of a worker testing module that stood independently from the rest of the result processing module. Gains in efficiency can be made for large scale efforts by bringing these two components back together, and having a more normalized database schema.

### 4.6. Earlier Architectures

The evolution of the system reflects changes in our understanding of how best to approach the challenges of using a crowdsourcing service like MTurk. Based on promising preliminary results (Rumshisky et al., 2009), our initial system had no quality control mechanisms in place.

This initial system only ran prototype experiments, and was set up to pay \$0.01 per HIT, with five sentence comparisons per HIT. On the back end, the initial system configuration was strictly object-oriented. All of the data structures required to keep track of experiments were formed as Python classes, and the experimental results were stored as serialized objects through the use of the Python pickling module. This approach to storing data was convenient because it did not require any traditional database design. However, the pickling module can become problematic as class definitions change, and it became apparent that a database representation would be needed to maintain the data with more rigor.

Experiments run under the initial system configuration yielded low quality data. The clusters generated by prototype experiments had very low f-scores when compared to the gold standard, and inter-annotator agreement was marginal.

Qualification requirements were introduced in the second iteration of the system as an attempt to raise worker quality without having to raise the cost of each

experiment. Two types of qualification requirements were tested: the locale qualification that is publicly available on MTurk, and a custom built qualification that required users to take a sample test to measure performance at the basic task.

Amazon's built in locale qualification allows requesters to specify a single country of origin for workers. Only workers in this country are qualified to work on these HITs and nobody else can access them. This qualification is granted automatically to workers based on the billing address they provide for their bank account to receive payment through MTurk. In contrast, the custom qualification that was tested in the second iteration of the system required workers to complete a small sample of the task before being granted access to the HITs. Workers who did not perform well on the sample were to be barred from accessing the rest of the HITs, allowing the qualification test to act as a first level filter. However, no workers from any locales attempted to complete the test, and this methodology was quickly abandoned.

Based on the assumption that the noisy data returned in the previous system test were being generated by workers from non-English speaking countries, the second round of experiments independently tested the impact of restricting locale to just the US or India (the two largest contributors of workers to the MTurk marketplace). These tests yielded several interesting findings. In the experiment where locale was restricted to the US, inter-annotator agreement increased significantly, but worker response time proportionately decreased. Worker interest was far too low to obtain results that could be compared to the gold standard. In contrast, the

prototype experiments with locale restricted to India obtained results very rapidly, but these results did not show any improvement in f-score compared to the results from the initial system configuration.

These findings led to a shift in focus driven by the question of how to attract the attention of high-quality workers based in the US. One of the issues of prototype style experiments is that only a small number of HITs get created initially. Although more HITs get dynamically created in later stages of a prototype experiment, to workers only a small number of HITs are available at any given time. From an economic perspective, learning how to perform a new task that pays a relatively small amount per task and only has a handful of tasks available is an inefficient use of time. It is fairly safe to assume that workers therefore tend to focus on tasks that potentially pay a lot of money per HIT with relatively few HITs available, as well as tasks that don't pay as much but have a huge pool of available HITs to complete.

The third iteration of the system tested the impact of running full pairwise experiments as compared to prototype experiments. They key difference between the two is that full pairwise experiments allow for a greater number of HITs to be posted all at once, which increases the amount of work (and therefore money) available to people interested in the task. In this configuration the price per HIT was kept at $0.01, and the locale was restricted to the US. This iteration also contained a modified back end that used the SQLite database management system and no longer relied on storing data as serialized Python objects. Results with this

configuration were slow to come back, but more complete and of a higher quality than any of the previous iterations of the system.

The last system configuration focused on ways to speed up the data collection process and improve overall worker quality. To speed up the data collection process, the price per HIT was adjusted to $0.03 based on budgetary constraints and worker feedback. To improve overall worker quality, it contained an implementation the worker quality control mechanisms described above in which worker error rates are algorithmically obtained compared to some gold standard data. This iteration also introduced two new qualifications that come built in to MTurk, allowing requesters to screen workers based on the number of HITs that workers have had approved on MTurk, and the overall percent of HITs approved.

CHAPTER 5

# Simulating Annotation Methodologies

## 5.1. Preliminaries

Data for only two experimental types was actually collected on MTurk during the course of this research. Prototype experiments are described in more detail below, but they can be described briefly as follows: A prototype experiment consists of a series of annotation rounds in which word sense clusters are built up after each round. At the beginning of a round, a prototype sentence is selected from the list of all unclustered sentences. Every other unclustered sentence is then compared to that sentence. After all of the comparisons have been made for a round, every sentence that was judged as similar to the prototype gets put in a sense cluster with the prototype sentence. This process repeats multiple times until all sentences have been put in a cluster.

Full pairwise experiments do not attempt to generate hard clusters like prototype experiments. Instead, judgments between every possible pair of sentences are collected, resulting in a fully saturated graph. In a full pairwise experiment, sense clusters are generated after all the data has been collected by running the Markov clustering algorithm on the resulting graph.

A major advantage to collecting full pairwise annotations is that the data allows for many other experimental methodologies to be tested. The goal of testing these different methodologies is to explore cheap and effective data collection alternatives that need only collect a subset of edges to emulate the results obtainable from a fully saturated graph.

By convention we refer to the testing of an experiment type based on full pairwise data a simulation. Simulations allow for proof of concept testing of different annotation methodologies under ideal conditions, cutting out much of the messiness required to actually build and test a working system of the methodology on MTurk.

In a simulation we imagine starting without any worker judgments. Depending on the experiment type being tested, we then observe worker judgments from a full pairwise graph in a systematic way that simulates what would happen during the experiment under real conditions, as if it were actually running on MTurk. This is possible because a full pairwise graph contains a comprehensive set of judgments. The resulting simulated experiment will contain a subset of edges observed from the full pairwise graph, and thus may have a dramatically different type of structure.

## 5.2. Prototype Simulations

Earlier we introduced prototype experiments. In this annotation methodology, data is collected in a series of rounds, with each round resulting in a distinct sense cluster. For the first round, a sentence is picked at random from all sentences,

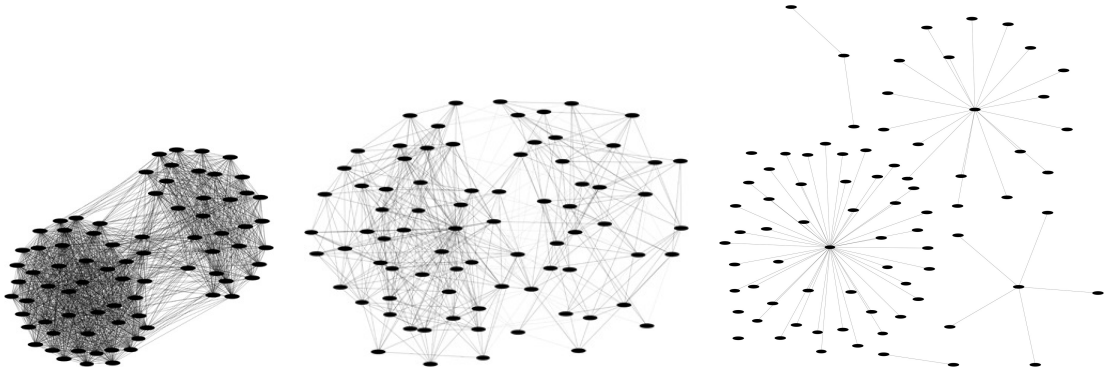

Figure 5.1. Different graphical representations for the verb rain. (Left) The full pairwise graph from which simulations are run. (Middle) A structure preserving simulation. (Right) A star shaped prototype simulation. All three graphs are drawn from the same data.

and every other sentences gets compared to this prototype sentence. At the end of the round, every sentence that was judged as similar to the prototype sentence is grouped in a cluster with the prototype. In the next round, the process is repeated with the remaining unclustered sentences. When every sentence has been placed in a cluster, the experiment is complete.

The term prototype is used here because the hope is that the sentence randomly selected at the beginning of each round employs a prototypical usage of some sense of the word we are trying to disambiguate. This may not be the case–the random sentence may contain a very unusual or highly ambiguous usage of the word in consideration. In the prototype methodology, at the beginning of each round after the prototype sentence has been selected, it needs to be tested to ensure that it has a clear usage compared to other sentences.

Prototype simulations from full pairwise data follow the basic structure outlined above. Below is psuedocode making explicit how to perform a prototype simulation from full pairwise data.

**Input:** A full pairwise graph for a single word.

**Output:** A list of clusters corresponding to different word senses.

**The prototype simulation process:**

While there are any remaining unclustered sentences:

**Pick Prototype:**

Randomly pick a prototype from the list of unclustered sentences

**Get Test Comparisons:**

Observe edges between the selected prototype and $n$ number of randomly selected sentences ($n$ is typically set to be only a fraction of the total number of unclustered sentences)

**Test Prototype Quality:**

Based on the edges observed above, determine if the prototype selected is unclear by seeing how varied the judgments are and how many unclear judgments were made.

**Unclear Prototype:** If the prototype is unclear, mark it for future reference as a poor prototype so that it doesnt get selected again and then go back to the Pick Prototype step

**Okay Prototype:** If the prototype is okay, observe edges between the prototype and all remaining unclustered sentences.

**Build Cluster (Sense):** Initialize a cluster for the round by placing the prototype sentence as the first item in the cluster. Every unclustered sentence that was judged as similar to the prototype by workers on MTurk gets added to the cluster for the round.

**Repeat:** With the remaining unclustered sentences after the step above, go back to the pick prototype step above and repeat until all sentences have been placed in a cluster.

The prototype methodology has several benefits over collecting full pairwise data. It significantly cheaper and in some instances has been shown to produce higher quality clusters when compared to the gold standard. The trade off for these gains is a loss in information about closely related or overlapping different senses are caused by sparse judgments collected between senses. Figure 5.2 is a good visualization of the general prototype methodology. Each distinct star shape is a cluster corresponding to a different sense of a word. In the middle of each star we have the prototype sentence, which gets compared to every other unclustered sentence.

## 5.3. Structure Preserving Adaptive Simulation

The prototype methodology may be good for inexpensively producing accurate sense distinctions, however, the resulting graph structure is highly segregated and does not contain a great deal of information about distances (and overlap) between senses that we initially set out to capture.

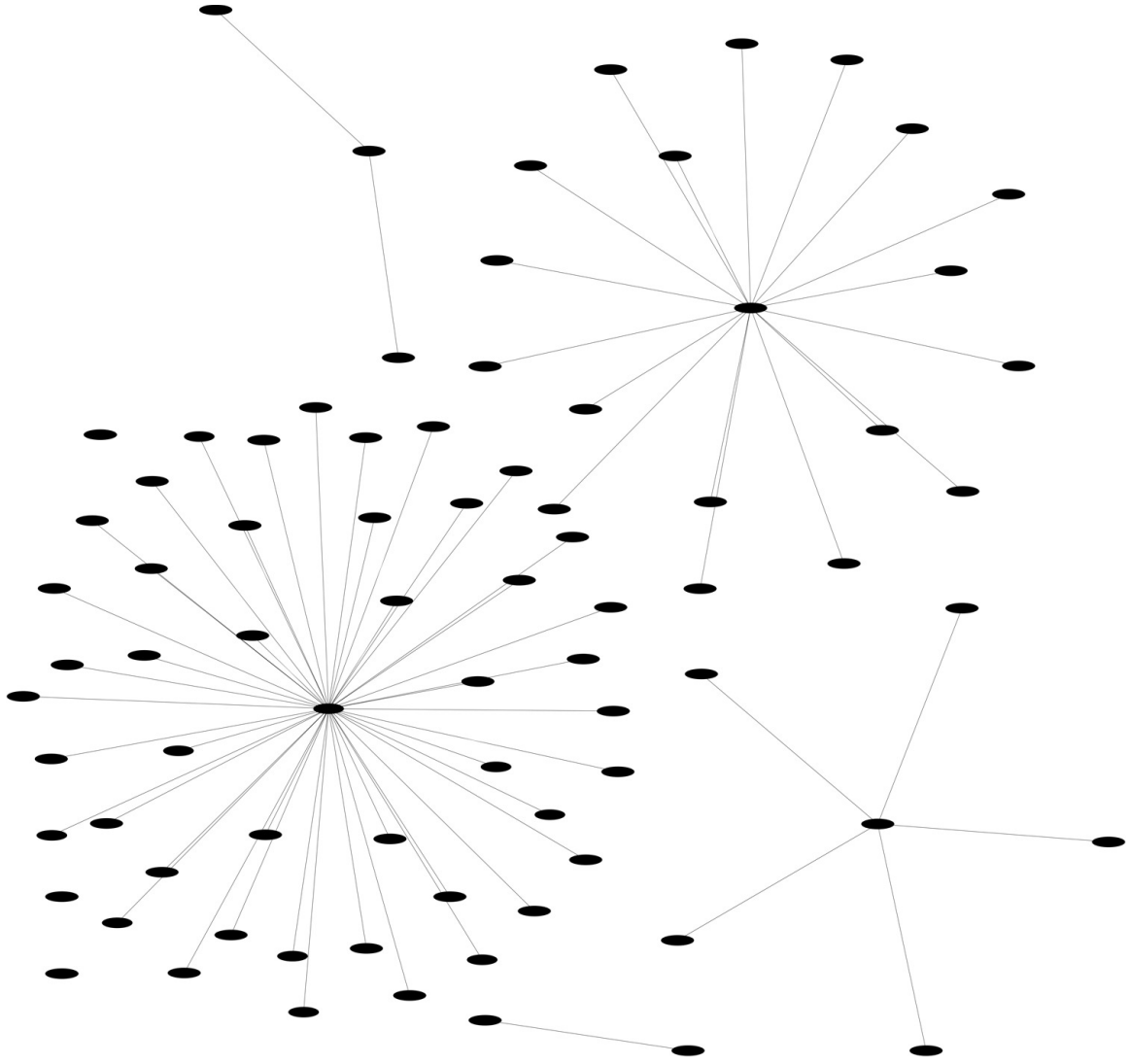

Figure 5.2. A prototype simulation generated from a full pairwise graph for the verb rain.

An experimental methodology is needed that allows the system to selectively sample a relatively small subset of edges creating a partially saturated graph that preserves the structure exhibited in a complete graph (i.e. a wireframe). One

group of methodologies for doing this involves iteratively collecting edges based on a heuristic about which edges will be most informative to observe. We call these methodologies adaptive because after each iteration when a sample of edges has been observed observed, a new set of edges to observe is adaptively selected based on the current structure of the graph.

For this thesis one simple structure preserving adaptive edge selection algorithm has been tested via simulation, and others are proposed later on. The structure preserving adaptive simulation (SPAS) runs as follows:

**Input:** A full pairwise graph of $n$ sentences with similarity judgments between every node; $r$, an integer defining the initial level of saturation in the graph (when $r == n$, saturation of the initial graph is 100% and the algorithm is exited).

**Output:** A pruned graph similar in structure to the input graph.

**The simulation process:**

**Initialize Graph:**

Initialize the graph by looping over all $n$ sentences, and for each one randomly select $r$ comparison sentences. Observe the comparisons, resulting in $n$ x $r$ initial edges in the graph.

Iterating $i$ times (or until convergence), improve the graph as follows:

**Generate Clusters:**

Run a clustering algorithm on the graph to obtain a set of hard clusters

**Compute Cluster Mediods:**

For every cluster compute the cluster mediod, the central most object in the cluster with minimal average dissimilarity to every other object (i.e. the sentence closest to the cluster centroid). At this step sentences are necessarily converted to vectors using a bag of words representation for each sentence (with lemmatization). Similarity is measured via cosine distances between vectors.

**Compare to Nearest Mediods:**

Loop over all n sentences and compare each sentence to the cluster centroid that is closest.

**Repeat:**

Return to the cluster generation step, re-clustering the graph based on the newly collected edges.

This structure preserving approach has several advantages over the prototype methodology, and can easily be modified to obtain different characteristics in the resulting graph. While data collection efforts under this methodology are comparable in price to the prototype methodology, unlike prototype simulations, the resulting graph is highly interconnected. This allows for sense clusters to be cut out of it at different levels of granularity, and provides more detailed information about how similar or overlapping two senses may be.

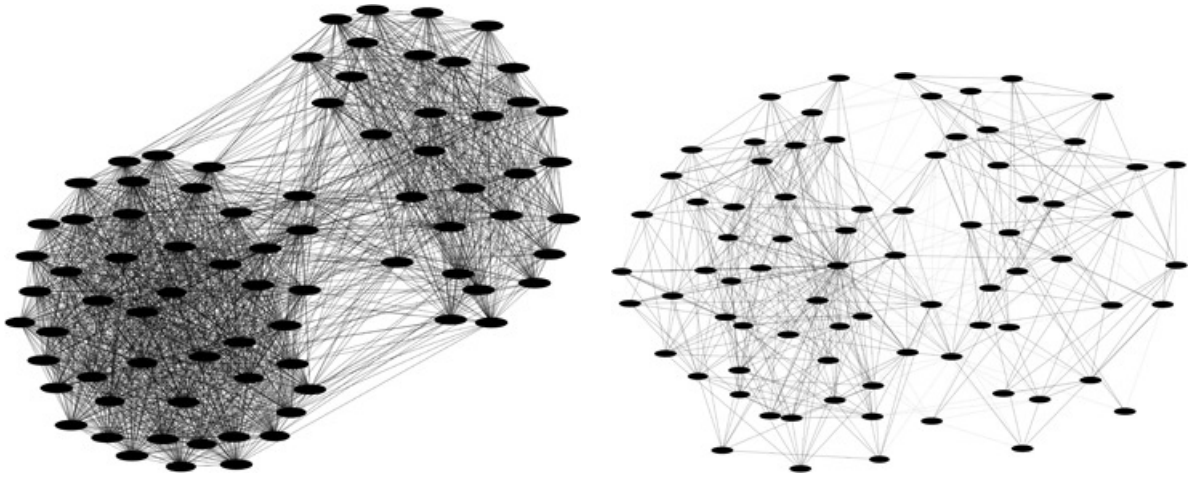

Figure 5.3. A side by side comparison of the full pairwise graph for rain (left) with the graph produced after running a structure preserving simulation on the same data (right). Both graphs exhibit roughly the same shape, but the structure preserving simulation features significantly fewer edges.

Figure 5.3 exhibits the end result of running a structure preserving adaptive simulation. While the simulation achieves the stated goal of approximately preserving the structure of a full pairwise graph using only a fraction of the edges, this methodology would benefit from a more rigorous and precise mathematical model for predicting which edges are expected to produce the highest information gain in the overall graph. In addition to this, the bag of words vector representation used to compute cluster centroids is fairly crude and could be enhanced with a more content rich sentence representation like topic modeling using LDA.

CHAPTER 6

# Results and Discussion

## 6.1. Early Prototype Experiments

Table 6.1 shows the results of early prototype experiments. These results underscore some of the technical challenges involved in using MTurk without proper quality control mechanisms in place.

Low quality results with three initial experiments (shut, open and swell) led to testing of the system with certain restrictions in place. When locale was restricted to US (lose, November), kappa and actual agreement scores jumped significantly, but the amount of time it took for workers to complete HITs was significantly longer. These findings are in line with other research indicating that more than half of the workforce on MTurk may be from outside of the US (Ipeirotis, 2010).

When locale was restricted to India (lose*, November), a high number of spammers flooded the system with low quality data. This experiment featured a new prototype quality testing mechanism that allowed for significantly more HITs to be posted at initialization, which increased the experiment's position in the HIT search space on Amazon, and led to significantly higher participation (and spamming). In contrast, when workers were presented with a custom qualification task that required them to answer a few questions before being allowed to complete the

| **Verb** | **shut** | **open** | **swell** | **lose** | **lose** † | **lose*** |
|---|---|---|---|---|---|---|
| **Start Date** | Sept | Sept | Oct | Nov | Nov | Nov |
| **Source** | CPA | ON | CPA | ON | ON | ON |
| **# of Sentences** | 150 | 150 | 150 | 150 | 150 | 150 |
| **Cost per HIT** ($) | 0.01 | 0.01 | 0.01 | 0.02 | 0.02 | 0.02 |
| **Comparisons per HIT** | 5 | 5 | 4 | 5 | 5 | 5 |
| **# of Annotators** | 5 | 5 | 5 | 5 | 5 | 5 |
| **Locale Restriction** | None | None | None | US | None | IN |
| **Max Error Rate** | None | None | None | None | None | None |
| **# of HITs created** | 68 | 95 | 89 | 30 | 30 | 222 |
| **Experiment completed** | Yes | Yes | Yes | No | No | No |
| **Total Cost ($)** | 3.40 | 4.75 | 4.45 | 2.96 | 0.00 | 22.20 |
| **Total # of Judgments** | 1700 | 2375 | 1780 | 725 | 0 | 5550 |
| **Approximate Runtime** | 2 Weeks | 2 Weeks | 2 Weeks | 3 Weeks | 1 Week | 1 Day |
| **Kappa** | 0.17 | 0.12 | 0.08 | 0.17 | – | 0.04 |
| **Actual Agreement** | 0.58 | 0.50 | 0.51 | 0.76 | – | 0.50 |
| **F-score** | 0.27 | 0.41 | 0.36 | – | – | – |

Table 6.1. Results for early prototype experiments posted to MTurk. († This experiment was run with a custom made qualification requirement that no workers completed.) (* This experiment was run with a different prototype quality testing configuration that allowed for significantly more HITs to be posted initially, leading to faster completion time.)

HITs for money (lose†, November), participation dropped to zero. The number of HITs that workers have immediately available to work on greatly impacts experiment completion time. This finding shifted experimental focus to collecting full pairwise data because significantly more HITs can be posted at initialization.

## 6.2. Full Pairwise Experiments

Table 6.2 shows the subsequent full pairwise experiments that have been run under different system configurations. While these experiments took slightly longer

| **Verb** | **shine** | **open** | **rain** | **realize** | **miss** |
|---|---|---|---|---|---|
| **Start Date** | Feb | Jan | Mar | Apr | Apr |
| **Source** | CPA | ON | CPA | ON | ON |
| **# of Sentences** | 84 | 100 | 83 | 80 | 78 |
| **Cost per HIT** ($) | 0.01 | 0.01 | 0.03 | 0.03 | 0.03 |
| **Comparisons per HIT** | 6 | 5 | 10 | 10 | 10 |
| **Locale Restriction** | None | US | US | US | US |
| **Max Error Rate** | 33% | 33% | 27% | 27% | 27% |
| **# of HITs created** | 616 | 1030 | 344 | 352 | 336 |
| **Experiment completed** | Yes | Yes | Yes | Yes | Yes |
| **Total Cost ($)** | 30.80 | 49.84 | 51.60 | 52.77 | 50.37 |
| **Total # of Judgments** | 18,480 | 25,750 | 17,200 | 15,800 | 15,020 |
| **Approximate Runtime** | 3 weeks | 5 weeks | 3 weeks | 3 weeks | 3 weeks |
| **Kappa** | 0.07 | 0.20 | 0.68 | 0.69 | 0.50 |
| **Actual Agreement** | 0.67 | 0.71 | 0.84 | 0.84 | 0.78 |

Table 6.2. A summary of system configuration settings for all full pairwise experiments posted to MTurk.

to complete than the early prototype experiments, they collected approximately 20 times more total judgments per experiment. As with the earlier prototype experiments, when locale was restricted to the US, the level of inter-annotator agreement rose significantly. Higher inter-annotator agreement was also observed when the system was reconfigured to pay workers $0.03 per 10 sentence comparisons as opposed to $0.01 for 5 sentence comparisons, resulting in a 50% increase in the average cost per judgment.

The F-scores reported in table 6.3 were obtained by iteratively running the Markov clustering algorithm (MCL) with different inflation values over fully saturated experimental graphs. The MCL algorithm can also be modified by varying

| Verb | shine | open | rain | realize | miss |
|---|---|---|---|---|---|
| **Best F-score** | 0.46 | 0.55 | 0.80 | 0.83 | 0.90 |
| **Inflation** | 7.40 | 4.03 | 6.44 | 1.50 | 6.44 |
| **Same weight** | 1.00 | 1.00 | 0.70 | 0.10 | 0.12 |
| **Diff weight** | -0.25 | -1.00 | -0.20 | -0.10 | -0.70 |
| **Unclear weight** | 0.20 | 0.25 | 0.20 | 0.025 | 0.025 |
| **Baseline** | 1.00 | 1.00 | 0.25 | 0.50 | 0.50 |

Table 6.3. A summary of the best clustering results for all full pairwise experiments. These results were obtained by running the Markov clustering algorithm on each experiment multiple times with a variety of different input parameters.

how edges are weighted in the input graph to the MCL algorithm. For the results presented in table 6.3, evaluation metrics were compared across not only inflation values but also a number of different edge weighting schemes. The edge weighting function takes as input a list of similarity scores between two sentences (with judgments from at least five annotators) and returns a scalar value where larger numbers indicate a greater level of similarity. The edge weighting function also takes as input the the following parameters used to vary how the graph gets represented to the MCL algorithm:

**Same weight:**

A weighting parameter indicating how much to add to a similarity score when two sentences are judged as *same* by a worker.

**Diff weight:**

A weighting parameter indicating how much to subtract from a similarity score when two sentences are judged as *different* by a worker.

**Unclear weight:**

A weighting parameter indicating how much to add or subtract to a similarity score when two sentences are judged as *unclear* by a worker.

**Baseline:**

The initial value assigned to an edge without any adjustment based on annotator input.

To produce a scalar edge weight, the function looks at each worker comparison between two sentences. When a worker judges two uses of the word in consideration as *same*, the value specified by the *same-weight* parameter gets added to the *baseline* parameter. This process continues for all (five) judgments provided between a pair of sentences; when a worker makes a *different* judgment the *diff-weight* amount is subtracted from the current edge weight.

It is important to note that scores can only be optimized a little bit by varying edge weighting parameters to best match the data. If the underlying quality of the data is noisy, no amount of optimization will produce useful results.

## 6.3. Simulated Experiments

### 6.3.1. Prototype Simulation Results

Table 6.4 shows results obtained by running 100 randomly initialized prototype simulations based on the full pairwise data for five words. Averages calculated across all 100 simulations for each word are presented.

| Verb | shine | open | rain | realize | miss |
|---|---|---|---|---|---|
| **Average F-score** | 0.44 | 0.38 | 0.76 | 0.86 | 0.80 |
| **Average Cost ($)** | 31.62 | 37.16 | 25.31 | 17.81 | 28.31 |
| **Average Kappa** | 0.07 | 0.20 | 0.43 | 0.67 | 0.52 |

Table 6.4. A summary of prototype simulation results averaged across 100 randomly initialized prototype simulations for every word.

One interesting feature of these results is the potential correlation between the quality of the data and the cost of collecting annotations for an experiment. When workers are performing at a high rate compared to the gold standard and with high inter-annotator agreement, the average cost of experiments appears to drop. This effect of lower quality simulations costing more may be a result of the fact that misclassification errors made early on in a prototype experiment have a tendency to propagate through the rest of the experiment, leading to more rounds of annotation and thus a higher total cost.

### 6.3.2. Structure Preserving Adaptive Simulation Results

Table 6.5 shows results obtained by running 10 randomly initialized structure preserving adaptive simulations (SPAS's). To ensure that comparisons reflect the quality of the underlying data and to prevent over-fitting, each simulation was run with the same parameters for generating clusters after each iteration. Because this methodology requires clusters to be generated after each iteration, and cluster centroids to be computed by projecting sentences into a high dimensional space, processing time is significantly longer than that of prototype simulations.

| Verb | shine | rain | realize | miss |
|---|---|---|---|---|
| **Average F-score** | 0.22 | 0.73 | 0.82 | 0.83 |
| **Average Cost ($)** | 17.98 | 30.39 | 30.75 | 29.96 |
| **Average Kappa** | 0.07 | 0.68 | 0.68 | 0.51 |

Table 6.5. A summary of structure preserving simulation results averaged across 10 randomly initialized trials for every word. To produce final evaluation scores, the graphs produced after running each simulation were cut into sense clusters using the MCL algorithm with the same edge weighting function and inflation values. Scores were averaged for each of the 10 simulations performed per word.

A comparison of the results between prototype simulations and SPAS simulations shows that both methodologies yield similar final quality data, but with dramatically different graphical structures.

### 6.4. Determining Data Quality Without a Gold Standard

The results presented in the sections above show a loose correlation between inter-annotator agreement and f-score. Where we see a high degree of inter-annotator agreement (measured by kappa scores), f-scores are generally high. However, inter-annotator agreement alone is not a perfect measurement of the overall quality of the data. This is due largely to the fact that sentences sitting on the boundary between two clearly distinguishable senses are expected to have a high degree of uncertainty and therefore low inter-annotator agreement.

Other metrics are needed to measure the overall quality of data in the absence of a gold standard. One obvious metric to use is the worker quality score described in section 4.4.2. For the more successful experiments, we found that the data

followed the Pareto principle: 80% of the data was provided by approximately 10% of workers who accessed our HITs, all of whom were necessarily high quality workers (due to the quality control measures in place).

Perhaps the best means of determining the quality of the underlying data is to manually compare sentences that get grouped together in sense clusters after an experiment has been run. Sentence groupings from early experiments demonstrate the baseline level of noise inherent to using MTurk without sufficient quality control mechanisms in place. Below are several example sentences that were clustered together in the same sense for the verb shine, which was an early full pairwise experiment with a low final f-score:

(1) . . . ev'n her face by kissing **shines** . . .

(2) . . . Crossman 's technique , dazzling for the time , **shines** through many of Stone 's records . . .

(3) . . . archaic faces seem to **shine** with an extrovert delight in life . . .

(4) . . . pasted patches of grass must **shine** in the distance like lighthouses . . .

Note that no distinction is being made between shining faces, which appear animated or bright, shining grass, which directs light, and a shining musical technique. In contrast, the results for our highest scoring experiment for the verb miss (f-score of 0.9) make the following distinctions (randomly sampled from each sense cluster):

**Sense Cluster 1:**

(1) . . . he never **missed** any classes . . .

(2) . . . not to **miss** the opportunity for bird-watching among the nearby estuaries . . .

(3) . . . the Saudi regime **missed** a fundamental element on which any political system is based . . .

(4) . . . conservative radio talk show hosts say the media is **missing** the story in Iraq . . .

**Sense Cluster 2:**

(1) The Enigma coding machine in question, . . . went **missing** last April from a glass case here at the Bletchley Park Museum.

(2) . . . there'll be two presidential candidates **missing** from the debate . . .

(3) . . . search teams looking for people who are still **missing** . . .

(4) . . . Under a microscope he could actually see that a bit of chromosome 13 was **missing** . . .

**Sense Cluster 3:**

(1) I **miss** you.

(2) What impressed me the most about him was that he really **missed** his parents.

(3) I want to tell them that I **miss** them a lot.

(4) "Frankly, I **missed** my family, " said Mr. Rosenblatt.

**Sense Cluster 4:**

(1) " What we are **missing**, " says author Yang Hsiao-yun, " is a way of bringing new meaning into the old tradition, a way of spending New Year that suits ordinary people. "

(2) . . . now that Congress has **missed** the legal deadline for meeting the Gramm-Rudman targets, the White House said it has returned to its original view that a capital-gains cut should be part of the deficit-reduction bill . . .

(3) That uh piece of foam **missed** the vehicle.

(4) I threw it at someone else but **missed**, and it hit her.

**Sense Cluster 5:**

(1) The bottles had **missed**.

CHAPTER 7

# Contributions and Future Work

## 7.1. Future Work

There are a number of possible improvements that could be made to the system with the aim of bringing down the waiting time and total cost of data collection while improving the overall quality of the results.

Being able to more precisely restrict worker locale could bring down the amount of time that it takes for HITs to complete. This is not currently possible through the MTurk API, but can be done through relying on worker self-reporting or by geolocating workers based on their IP addresses. Research indicates that workers are generally truthful when self-reporting their country of origin (Rand, 2011).

Several other simulation methodologies are worth exploring. For example, it is possible to design methodologies similar to the prototype methodology that produce tight clusters but also capture data about distances between clusters. Other methodologies might focus on only observing the edges that will be most informative in terms of placing a sentence into a sense cluster, based information-theoretic predictions. These numerous unexplored methodologies may provide more robust data at a fraction of the cost, and thus more experimentation on this front is needed and can be performed immediately using existing data.

The basic task that we proposed to workers can also be improved. Designing HITs that minimizes scrolling and reduce worker fatigue can have a big effect on the level of participation and the quality of data.

Most importantly, the utility of the resulting data set and partial lexicon created during the course of this research needs to be tested in broader applications (e.g. word sense disambiguation, information extraction, event tagging). We hope to show that a rich lexical resource can yield improvements in numerous other areas of the field.

## 7.2. Contributions

Our findings indicate that it is possible to tap into the linguistic intuition of non-expert native English speakers available on crowdsourcing services like Amazon Mechanical Turk, allowing for the development of an inexpensive and robust lexical resource. This adds to a growing body of work on collecting high quality linguistic annotations at a low cost through MTurk. Numerous iterations of our system were built and tested with only $300 to spend on data collection efforts. Furthermore, the findings made throughout the process of iteratively improving the system based on experimental results are instructive for future system design. We have uncovered a set of quality control mechanisms and best practices that dramatically increase the quality of our data and the speed at which it can be obtained, without a significant increase in cost.

Lastly, the data collected from experiments run under the optimal system configuration yield promising results. We have demonstrated that gold standard sense definitions can be almost fully recovered from non-expert annotations on MTurk.

APPENDIX A

# Database Schema

The following (SQL) table specifications make explicit the organization of data in the system.

## A.1. Experiments

The experiments table contains meta-data about the HITs that get created and judgments collected for different experiments.

**TABLE experiments:**

Contains meta-data about experiments and the system configuration they are being run under.

**word:** VARCHAR(200) NOT NULL

Experiment lemma for which data is to be collected.

**pos:** VARCHAR(200) NOT NULL

Part of speech of the word being tested (only verbs in this research).

**source:** VARCHAR(200) NOT NULL

The gold standard source of sentences and sense annotations (OntoNotes or CPA).

**exp-notes:** VARCHAR(200)

Supplemental information about an experiment.

**exp-type:** VARCHAR(50)

The type of experiment, i.e. the methodology being used to collect data and populate edges in an experimental graph (full pairwise or prototype).

**active:** BOOLEAN

True if results are still being collected for this experiment.

**complete:** BOOLEAN

True if all the data that needs to be collected given the experiment type has been collected.

**num-sentences:** INTEGER NOT NULL

How many sentences are being compared, i.e. how many nodes are in the experimental graph (typically between 80 - 150).

**num-assignments:** INTEGER NOT NULL

How many annotators are used to collect judgments.

**group-hits-by:** INTEGER NOT NULL

How many sentence comparisons are presented per HIT to workers (typically between 5 - 10).

**cost-per-assignment:** REAL NOT NULL

How much workers are paid per HIT.

**hit-title:** VARCHAR(400)

The title of the HIT presented to workers on Amazon.

**hit-summary:** TEXT

A brief summary of the task presented to workers on Amazon.

**lifetime:** INTEGER

How long until HITs expire and are no longer visible on MTurk.

**locale:** VARCHAR(2)

A two letter country code of where to restrict locale (e.g. US or IN). Empty string if no locale restriction is in place.

**approval-rate:** INTEGER

The *percent* of HITs previously submitted by a worker that must have been approved in order to qualify for the experiment HITs (typically 85%).

**num-hits-approved:** INTEGER

The *number* of HITs previously submitted by a worker that must have been approved in order to qualify for the experiment HITs (typically 200).

**hit-type:** VARCHAR(200)

An ID generated by MTurk to classify HITs for an experiment. All HITs for an experiment have the same HIT Type.

**total-cost:** REAL

How much money has been spent on an experiment thus far.

**ptest-threshold:** INTEGER NOT NULL

Minimum number of judgments between a prototype sentence and other sentences before the prototype can be tested for clarity (typically 150, 0 for full pairwise experiments which do not require prototype testing).

**exp-id:** INTEGER PRIMARY KEY

The unique identifier of an experiment.

**TABLE ExpSentences:**

Associated with every experiment is a list of sentences taken from a gold standard source. These sentences get compared to one another by workers on MTurk during the data collection phase of an experiment. They make up the nodes in an experimental graph.

**exp-id:** INTEGER NOT NULL

The unique ID of the experiment a sentence is placed.

**exp-ref:** INTEGER NOT NULL

An integer representing the sentence, used for compactness.

**sentence:** TEXT NOT NULL

The full text of the sentence.

**sense-num:** INTEGER

The gold standard sense number for the sentence.

**sentence-id:** INTEGER

A global sentence ID

**TABLE HitsCreated:**

Used to keep track of the HITs created for every experiment.

**HITId:** VARCHAR(200) NOT NULL

The unique ID for a HIT that gets posted to MTurk (this ID is returned by Amazon after a HIT gets created).

**exp-id:** INTEGER

The unique ID of the experiment for which a HIT was created.

**proto-id:** INTEGER

The sentence ID for the sentence displayed at the top of the HIT (i.e. the prototype sentence).

**comp-id:** INTEGER

The sentence ID the sentence which workers on MTurk are being ask to compare to the prototype sentence.

**spawned:** TIMESTAMP

The time when the HIT was posted to MTurk.

An experiment can be uniquely identified by its lemma, part of speech, the source of its gold standard annotations, the type of experiment that it is, and any additional experimental notes that the experiment creator might want to include to aid in processing. As was discussed earlier, when HITs are posted to MTurk they are annotated with this unique identifier so that when workers complete HITs, the data can be properly sorted in the database. Also associated with each individual

experiment is the set of sentences (each containing an instance the experimental word) to be compared.

Introduced in the schema for the experiments table is the notion of having different types of experiments, each of which can be distinguished by the structure of their resultant graphs and the way in which comparisons between sentences are collected from workers. The most general type of experiment is a full pairwise experiment, named such because a full pairwise set of comparisons is collected, which leads to a fully saturated graph.

The full pairwise experiment serves as the foundational model for all other experiment types. With the expressed goals of creating a quantitatively rich lexical resource, full pairwise experiments are ideal because they contain the most information. However, because the number of comparisons that need to be collected is quadratic with respect to the number of sentences being compared, in practice the full pairwise set up is prohibitively expensive.

Thus, every other experiment type should be thought of as an attempt to reproduce a full pairwise graph (at partial saturation) by intelligently selecting a subset of edges to observe, thereby reducing the overall cost of data collection.

Initially, this system was set up to test full pairwise experiments in tandem with another experiment type that was hypothesized to produce favorable results. Eventually focus shifted to exclusively full pairwise experiments because of difficulty maintaining multiple experiment architectures and the realization that all

other experiment types could be simulated with full pairwise data. Different experimental setups are discussed in more detail in the post-processing section.

Finally, with each experiment we have a list of all the HITs that have been created and posted to MTurk.

## A.2. Charts

Associated with every experiment is a two-dimensional chart that holds the judgments that have been collected for an experiment. The indices in each chart represent sentences, and the cells in the chart contain a list of similarity judgments between pairs of sentences returned by workers on MTurk. This chart representation is interchangeable with the graph representation mentioned earlier in which sentences as nodes and edges are similarity judgments.

**TABLE Charts:**

Holds the experimental results (i.e. worker judgments) that get returned from MTurk.

**exp-id:** INTEGER

The ID of the experiment to which the judgment is relevant.

**proto-id:** INTEGER

The sentence ID for the sentence displayed at the top of the HIT (i.e. the prototype sentence).

**comp-id:** INTEGER

The sentence ID the sentence which workers on MTurk are being ask to compare to the prototype sentence.

**HITId:** VARCHAR(200)

The unique ID for the HIT containing the comparison question.

**AssignmentId:** VARCHAR(200)

The unique ID for the HIT results returned by a worker.

**WorkerId:** VARCHAR(200)

The unique ID of the worker who made the judgment. Amazon assigns this value to workers when they sign up on MTurk.

**similarity:** INTEGER NOT NULL

The final worker judgment on whether or not the proto-id sentence and the comp-id sentence use the experimental word with the same meaning or not (1 = same, 0 = different, -1 = unclear).

**bad-result:** BOOLEAN

True if worker who provided this similarity judgment is a spammer. Used to filter out results from low quality workers.

The judgments that get added to a chart come from worker assignments. As a reminder, Amazon makes a distinction between HITs and assignments. When a worker completes a HIT, their assignment of that HIT becomes available to the requestor for reviewing. Thus, the system uses the assignments table defined below

as a mechanism for tracking which workers completed which assignments for which HITs.

**TABLE assignments:**

For tracking worker assignments completed for different HITs. An assignment contains the results for a single annotator for a given HIT (HITs typically have five assignments = five different annotators).

**AssignmentId:** VARCHAR(200) PRIMARY KEY

The unique ID of a worker's response to a particular HIT.

**WorkerId:** VARCHAR(200) NOT NULL

The unique ID of the worker who made the judgment. Amazon assigns this value to workers when they sign up on MTurk.

**HITId:** VARCHAR(200) REFERENCES HitsCreated (HITId)

The unique ID for the HIT which the assignment corresponds to.

**exp-id:** INTEGER

The ID of the experiment to which the judgment is relevant.

**AssignmentStatus:** VARCHAR(200)

Indicates whether or not the assignment has been approved for payment on MTurk, rejected, or unprocessed.

**AcceptTime:** DATETIME NOT NULL

When the worker accepted the HIT and began working on it.

**SubmitTime:** DATETIME NOT NULL

When the worker completed the HIT and sent it back for processing.

### A.3. Worker Statistics

Lastly, we have a table for tracking meta-data about workers, used for quality control purposes.

**TABLE WorkerStats:**

Used for tracking worker quality across experiments.

**WorkerId:** VARCHAR(200) NOT NULL PRIMARY KEY

Unique worker id, as assigned by Amazon.

**ErrorRate:** REAL

Error rate of a worker vs gold standard data, determined by running the GetAnotherLabel worker quality assessment software.

**NumAnnotations:** INTEGER

How many comparisons a worker has made across all experiments.

**blocked:** BOOLEAN NOT NULL

True if worker error rate goes too high or if the worker is a spammer.

**LastBonus:** INTEGER DEFAULT 0

The number of annotations at which the last bonus was paid. Bonuses are typically paid out every 500 judgments.

**BonusAmount:** REAL DEFAULT 0.0

The total amount of money paid out to a worker in the form of bonuses.

Worker quality can be tracked across all experiments, however, this may result in the blocking of certain types of legitimate workers who perform well at disambiguating certain words, but who dont make the right distinctions for other words. For this reason, the system separates out the functionality of collecting and analyzing worker statistics in such a way that different experiments (or groups of experiments) can be plugged into distinct worker databases.

APPENDIX B

# Table of System Parameters

The following is a summary of the most commonly used parameters for adjusting the behavior of the data collection system.

| **Parameter** | **Description** | **Default** |
|---|---|---|
| approval rate | what percent of previously submitted HITs workers must have approved to be qualified for HITs to be created | 85 |
| bonus threshold | how many comparisons a worker must make before being eligible for a bonus | 500 |
| bonus error threshold | the maximum error rate above which a worker is no longer eligible for a bonus | 0.17 |
| bonus percent | the percent of total earnings paid out to workers qualified to receive bonuses | 10 |
| cost per assignment | the amount of money paid to a worker per approved assignment | $0.03 |
| lifetime | the duration (in seconds) before HITs posted to MTurk expire and can no longer be completed | 2 weeks |
| number of HITs approved | the total number of HITs previously submitted by a worker that have been approved | 200 |
| test locale | the country code specifying a locale to restrict HITs to (left blank for no locale restriction) | None |
| minimum number of assignments | the minimum number of assignments that need to be collected before worker error rates can be calculated | 25 |
| maximum error rate | the maximum error rate above which a workers assignments are rejected | 0.27 |
| number of annotators | the number of annotators that can simultaneously work on the same HIT | 5 |

Table B.1. A summary of commonly used system parameters.

# Bibliography

(n.d.).

Agirre, E., & Soroa, A. (2007, June). Semeval-2007 task 02: Evaluating word sense induction and discrimination systems. In *Proceedings of the fourth international workshop on semantic evaluations (semeval-2007)* (pp. 7–12). Prague, Czech Republic: Association for Computational Linguistics. Available from `http://www.aclweb.org/anthology/W/W07/W07-2002`

Akkaya, C., Conrad, A., Wiebe, J., & Mihalcea, R. (2010). Amazon mechanical turk for subjectivity word sense disambiguation. In *Proceedings of the naacl hlt 2010 workshop on creating speech and language data with amazon's mechanical turk* (pp. 195–203). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from `http://dl.acm.org/citation.cfm?id=1866696.1866727`

Anna Rumshisky, S. K., Nick Botchan, & Pustejovsky, J. (2012). Word sense inventories by non-experts. *Proceedings of the Conference on Language Resources and Evaluation (LREC 2012)*.

C., C.-B., & Dredze, M. (Eds.). (2010). *Csldamt '10: Proceedings of the naacl hlt 2010 workshop on creating speech and language data with amazon's mechanical turk.* Stroudsburg, PA, USA: Association for Computational Linguistics.

Denkowski, M., Al-Haj, H., & Lavie, A. (2010). Turker-assisted paraphrasing for english-arabic machine translation. In *Proceedings of the naacl hlt 2010 workshop on creating speech and language data with amazon's mechanical turk* (pp. 66–70). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from `http://dl.acm.org/citation.cfm?id=1866696.1866707`

Evanini, K., Higgins, D., & Zechner, K. (2010). Using amazon mechanical turk for transcription of non-native speech. In *Proceedings of the naacl hlt 2010 workshop on creating speech and language data with amazon's mechanical turk* (pp. 53–56). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from `http://dl.acm.org/citation.cfm?id=1866696.1866704`

Fellbaum, C. (Ed.). (1998). *Wordnet: an electronic lexical database.* MIT Press.

Fleiss, J., et al. (1971). Measuring nominal scale agreement among many raters. *Psychological Bulletin*, *76*(5), 378–382.

Gonzalo, J., Verdejo, F., Chugur, I., & Cigarrán, J. M. (1998). Indexing with wordnet synsets can improve text retrieval. *CoRR*, *cmp-lg/9808002*.

Gordon, J., Van Durme, B., & Schubert, L. K. (2010). Evaluation of commonsense knowledge with mechanical turk. In *Proceedings of the naacl hlt 2010 workshop on creating speech and language data with amazon's mechanical turk* (pp. 159–162). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from `http://dl.acm.org/citation.cfm?id=1866696.1866720`

Hanks, P. (2000). Do word meanings exist? *Computers and the Humanities*, *34*(1), 205–215.

Ipeirotis, P. (2010). Analyzing the amazon mechanical turk marketplace. *XRDS: Crossroads, The ACM Magazine for Students*, *17*(2), 16–21.

Ipeirotis, P., Provost, F., & Wang, J. (2010). Quality management on amazon mechanical turk. In *Proceedings of the acm sigkdd workshop on human computation* (pp. 64–67).

Moldovan, D., & Mihalcea, R. (2000, jan/feb). Using wordnet and lexical operators to improve internet searches. *Internet Computing, IEEE*, *4*(1), 34 -43.

Pustejovsky, J. (1995). *Generative Lexicon*. Cambridge (Mass.): MIT Press.

Rand, D. (2011). The promise of mechanical turk: How online labor markets can help theorists run behavioral experiments. *Journal of theoretical biology*.

Rumshisky, A. (2011). Crowdsourcing word sense definition. *Proceedings of 5th Linguistic Annotation Workshop, ACL HLT 2011*, 74.

Rumshisky, A., Moszkowicz, J., & Verhagen, M. (2009). The holy grail of sense definition: Creating a sense disambiguated corpus from scratch. In *Proceedings of 5th international conference on generative approaches to the lexicon (gl2009)*.

Snow, R., O'Connor, B., Jurafsky, D., & Ng, A. (2008). Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In *Proceedings of the conference on empirical methods in natural language processing* (pp. 254–263).

Snow, R., Prakash, S., Jurafsky, D., & Ng, A. (2007). Learning to merge word senses. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning* (pp. 1005–1014).

Tamuz, O., Liu, C., Belongie, S., Shamir, O., & Kalai, A. T. (2011). Adaptively learning the crowd kernel. *CoRR*, *abs/1105.1033*.

Van Dongen, S. (2000). A cluster algorithm for graphs. *Report-Information systems*(10), 1–40.

Zhao, Y., Karypis, G., & Fayyad, U. M. (2005). Hierarchical clustering algorithms for document datasets [journal]. , *10*, 141–168.