CASTLE: Crowd-Assisted System for Textual Labeling and Extraction

ABSTRACT

Temporary, will be rewritten.–SLG The need to automatically process and store large amounts of uncertain and imprecise machine learned data has necessitated the use of Probabilistic Databases (PDBs) which maintain and allow queries that carry a degree of uncertainty. An integral part of the data cleaning process is finding efficient ways to reduce this uncertainty. In this paper we propose the Crowd Assisted Machine Learning (CAMeL) paradigm which seeks to utilize crowdsourcing techniques such as Amazon Mechanical Turk to optimally improve the accuracy of learned data. This paradigm is implemented on top of a probabilistic database which we call CAMeL-DB. A subset of the automatically populated fields are converted into questions to be answered by the crowd. We demonstrate the efficacy of our approach on an information extraction problem consisting of automated segmentation of bibliographic citations, showing that a relatively small subset of questions can lead to a large boost in accuracy.

1 Introduction

Need to figure out where to put HIT maintenance and API information.–SLG

Will be modifying to keep overall picture in line with rest of paper.–SLG

The web is becoming an ever increasing expanse of information and knowledge. Unfortunately, the majority of this data is not easily manipulated or analyzed by computers. Granting structure to the trove of unstructured data for storage in a database is the key to efficient and complex searching, querying, and analysis. Traditionally it's been the job of humans to provide such metadata and structure, filling out the database by hand, but this is typically a slow and expensive process. Information Extraction is the method of performing this annotation automatically and at scale, rapidly increasing speed and dramatic lowering costs.

Consider the following example scientific citation.

[Citation]

Recognizing certain fields such as the title or author are simple tasks for most individuals, but represents a challenging problem for machines. One of the leading automated techniques employs the use of linear-chain conditional random fields (CRFs) [5], a generalization of hidden Markov models, for sequential tagging.

Reference further attempts at using CRF for IE.–SLG

While there are many advantages to be gained from automation, even the most start-of-the-art algorithms are not without error. There is a well known tradeoff [7] between the level of accuracy achieved by human processing and the speed and financial gains from machine processing.

Recently there has been increasing development of "human computation marketplaces" such as Amazon Mechanical Turk. Developing microtasks that can be distributed concurrently to thousands of people at once at reduced cost has greatly increased the utility of hiring human workers to do trivial tasks such as annotation, ranking, and searching on the internet. While significantly cheaper than hiring human experts, the cost of crowdsourcing is still much greater than processor a task using automated techniques. The key is being able to use both efficiently.

In this paper we introduce CASTLE, a system designed to take advantage of the strengths of both human and machine computation in a unified manner. The goal is to introduce humans only after machine algorithms have run, cleaning up and improving those elements of the output that are the most uncertain. Our specific notion of uncertainty is defined is expanded upon in Section **??**.

There are many challenges which need to be met in designing a system such as CASTLE. Uncertainty needs to managed and maintained throughout the database. The number of questions which represent the mapping from specific fields to the crowd should be minimized to control costs. Lastly, quality should be maintained even when dealing with a possibly noisy and conflicting crowd. CASTLE makes a number of contributions to address these challenges.

Uncertainty is inherent in the construction of the system. Probabilities associated with the machine learning results are stored in the base tables and manipulations of the data behave probabilistically according to their underlying distributions. There is a philosophical reflection to be had in the treatment of data this way. Decisions about the structure of data are always inferred, never truthed, and incoming evidence, be it from additional learning algorithms, human experts, or crowdsourced responses, always has the ability to update these decision processes.

Even with the advent of the crowd, contracting an entire data set out could still prove costly. Only selecting those fields for which there is a reasonable enough assumption of incorrectness would drastically reduce the cost and allow those examples "easy enough" to be done by mechanical methods to be done swiftly and cheap. CASTLE uses information theory in a manner similar to uncertainty sampling in active learning for this selection process. It also has the power to recognize redundancy and map multiple fields into

SIGMOD '13 New York, New York USA

a single question.

The greatest drawback to using a crowd of non-experts is the noisiness of the response. The economics of the Amazon Mechanical Turk marketplace provides little incentive for a worker to submit high quality work and weeding out the response of lazy and malicious workers is an area of active research in the crowdsourcing community. One of the standard techniques is to take a majority vote among a collection of workers, but such a deterministic approach is still highly susceptible if all workers are of low quality and removes possible controversy or conflict over challenging questions. Since CASTLE is a probabilistic database, we use Bayesian and Dempster-Shafer approaches to integrate responses probabilistically. As far we know, we are the first to pursue such an integration among crowd responses.

Success and utility of the system is envinced in two ways: decreasing of the number of HITs needed to clean a database and increasing of the accuracy of worker results. Using a combination of high entropy selection and multi-token clustering, we were able to reduce the number of HITs by many orders of magnitude compared to a random baseline. Also, in addition to being more informative in terms of overall crowd response, our probabilistic integration method using Dempster-Shafer exhibited a XX% gain in accuracy over its deterministic counterpart, majority voting.

This paper is organized as follows. Section **??** chronicles an overview of the system and its various probabilistic components. Selection and clustering of fields is discussed in Section **??**. HIT management from within the system is contained in Section **??**. We discuss our approach to probabilistic integration in Section **??**, while Section **??** contains our experiments. Finally, Section **??** contains the conclusion and Section **??** our future work.

2 Related Work

Might switch related work and background or move related work to back–SLG Discuss previous attempts at citation IE (esp. by McCallum)

Identify various methods of data cleaning/integration

Compare selection process to that used in active learning

Introduce crowdsourcing and related quality control efforts

3 Background

3.1 Probabilistic Databases

3.2 Conditional Random Fields (CRF)

3.3 Inference Queries over a CRF Model

3.4 Uncertainty Sampling

3.5 Crowdsourcing

4 System Overview

This section describes the basic setup of the system. We extend the in-database CRF model of Wang et al.[10], giving an overview of the original data model and then discussing our extensions to handle crowd-inserted data and newly inferred evidence. Afterwards we explore the various operators used to manipulate the data and perform functions such as selecting uncertain tokens, pushing to the crowd, aggregating the result, and integrating new evidence into the final query results.

4.1 Data Model

We treat unstructured text as consisting of a set of documents or text-strings $\mathcal{D}$. Each document $d \in \mathcal{D}$ has substructure comprising a set of tokens t_i^d, where $i \in \{1, ..., N\}$ and N is the length of the string. Specific documents d are identified by a unique identifier (docID) and tokens t_i^d are identified by their associated docID and their position i within d.

TOKENTBL and FACTORTBL below are identical to the versions found in [10]. In order to provide an interface for human-corrected answers we introduce CROWDPOSTTBL for keeping track of questions posted to the crowd, CROWDANSTBL for storing responses, and EVIDENCETBL comprising aggregated evidence used to constrain the inference process at query time.

Token Table: The token table TOKENTBL is an incomplete relation R in $\mathcal{DB}^{\mathcal{P}}$, which stores text-strings as relations in CASTLE. The structure is similar to that found in inverted file systems commonly used in information retrieval. As previously mentioned the primary key for each token is the docID and position (pos) attributes. The token also contains one probabilistic attribute—label^p. Its values are left initially empty and are inferred from a combination of the learned machine model and crowd-provided labels.

TOKENTBL(docID, pos, token, label^p)

Factor Table: The FACTORTBL allows for a direct computation of the possible "worlds" of TOKENTBL from within CASTLE according to the CRF model. It is a materialization of the *factors* used to calculate edge potentials $\varphi[y_i, y_{i-1}|x_{i-1}]$ to determine the most likely labeling of the tokens in corpus $\mathcal{D}$. In the CRF model, these factors defining the correlation between x_i, y_i, and y_{i-1} are the weighted sum of feature functions: $\varphi[y_i, y_{i-1}|x_i] = \sum_{k=1}^{K} \lambda_k f_k(y_i, y_{i-1}, x_i$

These features are typically binary and denote the presence or absence of x_i, y_i, and y_{i-1} appearing together in the model. The weights are learned from training and provide a score to the set of correlated variables. FACTORTBL contains each triple along with its associated score.

FACTORTBL(token, prevLabel, label, score)

Crowd Post Table: The set of tokens $\mathcal{T}$ that are selected to be hand-labeled need a means of mapping to and from the Amazon Mechanical Turk service. Each submitted HIT is given a unique HITID by Amazon, which is stored in CROWDPOSTTBL to maintain token identities upon retrieval for each $t \in \mathcal{T}$. Since many tokens are mapped into a single HIT, as we describe later in Section **??**, the table also includes the question position (HITpos).

CROWDPOSTTBL(docID, pos, HITID, HITpos)

Crowd Answer Table: The crowd's responsibility is to provide labels from the set $\mathcal{L}$ for all tokens pushed to the AMT service. Individual labels $l \in \mathcal{L}$ retrieved for each HIT are stored in CROWDANSTBL along with a set of identifiers for the HIT (HITID & HITpos) and the Turker herself (workerID). Performing a JOIN with CROWDPOSTTBL produces a set of token-label pairs. However, since question redundancy is one method of quality control there may be more than one label per token and the answers must be aggregated before they may be used. This is discussed more in Section **??**.

CROWDANSTBL(HITID, HITpos, workerID, label)

Evidence Table: When the Viterbi process is run, it consults the evidence table EVIDENCETBL and constrains the result accordingly. It has the same format as TOKENTBL, but while the latter is constructed purely from the extracted ML output, the former is

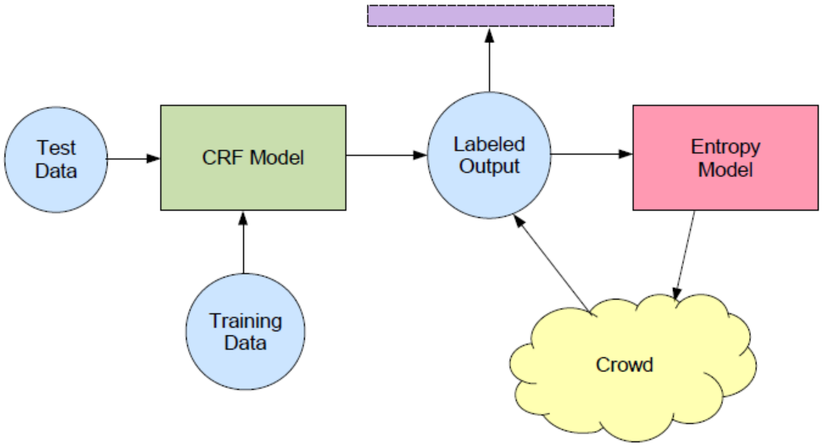


Figure 1: Architecture of the CASTLE system.

drawn from additional sources and take precedence over the extraction. The probabilistic attribute, label^p, represents a probabilistic aggregation of the wisdom of the crowd. Section **??** contains an explanation of the entire integration process.

$$\text{EVIDENCETBL(docID, pos, token, label}^p\text{)}$$

4.2 CASTLE Operations

The operations of CASTLE are defined over a set of User-Defined Functions (UDF) on the relational tables that express the full fuctionality of the system. Apart from traditional insert, remove, and query operations, CASTLE has a number of more complex functions that update the internal state of the system. These generally fall into functions supporting one of three broader operations: selection, aggregation, and integration.

Selection of uncertain tokens is based on information theoretic uncertainty sampling. CASTLE has functionality for computing marginal distributions over token labelings and the associated entropy of those distributions. A question queue is created for pushing tokens to AMT ranked by highest entropy. System-specific optimizations include clustering over duplicated tokens to prevent redundancy in question submissions and restricting to one token per document d.

Aggregation is used as a means of preserving the quality of answers. The gold standard is to increase the number of workers that answer each question and take a majority vote among their responses. In our studies we've found that a Bayesian approach based on perceived Turker quality produces consistently better results and has the advantage of being probabilistic. This technique is implemented in CASTLE along with the sub-function to calculate worker quality.

Finally, integration allows the database to use the crowd-aggregated result to improve its own extraction results. The vanilla Viterbi is replaced by a Constrained Viterbi that modifies its path according to feedback in its evidence table. The user has the ability to place a threshold on the entropy of probabilistic evidence values to ensure only high quality, trusted feedback is actually used.

The remainder of this paper will be spent fleshing out the procedures behind selection, aggregation, and integration. Specific UDFs will be reviewed for each section, while full pseudo-code and SQL statements can be found in the Appendix.

5 Selection

Move to intro from here—-SLG Current implementations of crowdsourcing in databases such as CrowdDB [3] and Qurk [6] have focused primarily on using human computation at the query processing level, enabling human workers to fill in missing tables when the data is queried. The query itself allocates on-line which entries should be modified by humans. CASTLE contrasts with this methodology by pre-processing the crowdsourcing portions offline.

The problem that results is in which entries should be sent to the crowd for modification. The underlying data model of CASTLE is a database of tokens. We can represent each token in terms of a question posted to Mechanical Turk. Such questions provide a token and allow workers to supply the true label of that token. Given that large scale databases may contain millions of token entries, asking any type of large subset can become prohibitively expensive. For instance, asking 5 Turkers per question at $0.01 apiece, 100,000 tokens would still cost $5,000. Therefore, it would help to limit the number of questions needed to produce a sizeable gain in accuracy of the database.–to here–SLG

In this section, we discuss some optimizations that seek to minimize the number of questions one needs to ask the crowd. First we discuss methods to constrain the token space, such as limiting only token per document and clustering similar documents into a single question. Then we describe how to order the remaining space so that given a fixed budget of questions we can optimize the number of cleaned entries using information theory. We term the entire selection process InfoSelect and the full algorithm is displayed in Figure 1.

5.1 Basic Entropy-based Algorithm

The first approach to optimizing the set of questions is to determine which ones give the highest quality information. The selection problem we encounter can be framed in terms of a *Total Utility*

Algorithm 1: InfoSelect

```
input  : Full database T of tokens
output: Reduced set S of maximum information tokens
Initialize hash map H;
Initialize cluster set C;
//Retain only max entropy tokens from
each citation;
foreach t ∈ T do
    if H.containsKey(t.docID) then
        currentToken ← H(t.docID);
        if currentToken.entropy < t.entropy then
            H(t.docID) ← t;
    else
        H(t.docID) ← t;
//Cluster tokens with similar properties;
Load all tokens in H into queue Q;
foreach t ∈ Q do
    foreach cluster c ∈ C do
        if c.text = t.text &
        c.label = t.label &
        c.prevLabel = t.prevLabel &
        c.postLabel = t.postLabel then
            Add t to cluster c;
            c.totalEntropy ← c.totalEntropy +t.totalEntropy;
    if t not added to a cluster then
        Initialize new cluster c;
        c.text ← t.text;
        c.label ← t.label;
        c.prevLabel ← t.prevLabel;
        c.postLabel ← t.postLabel;
        Add c to cluster set C;
SORT clusters c ∈ C by c.totalEntropy;
```

Function (TUF) that quantizes the total uncertainty in the database. Selecting tokens for labeling by the crowd corresponds to a reduction of the TUF and we seek to select those tokens that do this maximally. While strictly speaking the TUF can be any function of the probabilistic content of each token, we model our function on the total entropy of the system. For an individual document, this is the entropy over the space of possible labelings **s**:

$$H(d^i) = \sum_{\mathbf{s}} p^i(\mathbf{s}) log(p^i(\mathbf{s})) \quad (1)$$

Since all documents in the database are independent, the total entropy is equivalent to the sum of the entropies of each document $\sum_i H(d^i)$. For a document of length N in a label space of size L there are N^L possible values that **s** may take, making direct calculation intractable for any nontrivial task. We can bypass this issue by remembering that absolute values of the system entropy are not needed and we care only about relative changes in response to selecting one token over another.

We consider our selection problem to be one of partitioning the set of all tokens T into those selected for submission and those not. Let X be the set of tokens pushed to the crowd and Y be the remaining set of tokens such that $Y = T - X$. For a fixed budget, we seek to select X such that $H(Y|X)$ is minimized. The equation for conditional entropy states:

$$H(Y|X) = H(X,Y) - H(X) \quad (2)$$

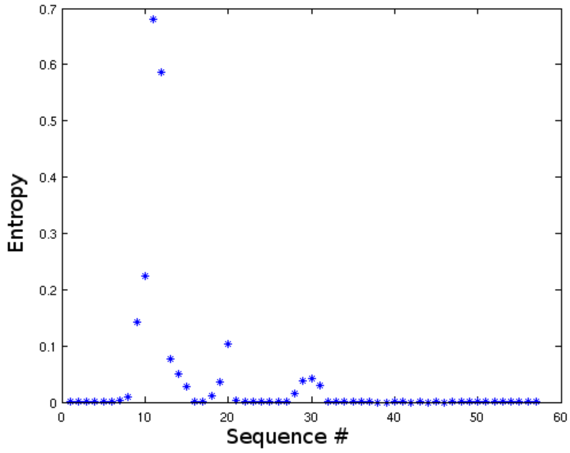


Figure 2: Typical entropy distribution for a document in CASTLE

The first part of the right hand side, $H(X,Y)$, is a constant since $X \cup Y = T$. Thus minimizing $H(Y|X)$ is equivalent to maximizing $H(X)$, the entropy of the selected tokens. If selection is done on-line one token at a time, $H(X)$ is the marginal entropy of an individual token and top-k selection is equivalent to selecting the top-k marginal entropies from T. Explicitly, the marginal entropy for token i is:

$$H^i = -\sum_{j=1}^{L} p_j^i log(p_j^i) \quad (3)$$

where the p_j is the marginal probability of label j being the correct label. Marginal probabilities are calculated using the CRF-variant of the Forward-Backward algorithm []. The marginal entropy provides a formulation for a token's individual certainty or uncertainty in its label. Compare the distributions in Figure 5.1. When the distribution is fairly peaked and we are certain of an answer, the entropy tends to be fairly low. A more uniform distribution, however, leads to a higher entropy.

5.2 Reducing the Space of Questions

5.2.1 Constraining Tokens Per Document

While we may sort all of the tokens in the database by entropy and select the top-k, this may lead to less than optimal results. Individual tokens are not independent. Modifying the probabilistic label attribute of one token in a document may invoke additional corrections when the inference algorithm is re-run. We discuss this constrained inference idea further in the context of data integration in Section **??**. Suffice to say, because we can't anticipate the expected gain from each individual token, we choose not to risk the redundancy of batching multiple high entropy tokens from the same document.

The motivation for this choice can be found in Fig **??**, which shows the entropy distribution for a typical document. The highest entropy values appear in isolated neighborhoods which correspond to the segmentation boundaries. Since constrained inference primarily supplies additional correction the neighborhoods of constrained tokens, the probability of two selected tokens sharing the same "correction window" is high.

Thus the strategy implemented is to select only a single token from each document, that token being the one with the highest entropy. The explicit process can be found in Figure **??** and uses a simple hash map with a docID key to assert only one token per docID is allowed. All tokens are cycled through in a linear fashion. If the current docID is not represented in the map, we add it, while if it is represented we only add it if it's entropy is higher than the token currently in the map. With each document being associated with its highest entropy token, for the remainder of this discussion we refer to selection of a token and selection of document interchangeably.

The existence of such a "correction window" as mentioned above brings up an interesting tradeoff that should be considered. When the two differ, is it better to select a single highly uncertain token or a token at the center of a high entropy neighborhood, none of whose entropy is as high as the first? As an additional entropy metric for selection, we introduce the concept of "neighborhood entropy", that is the entropy associated with a token and its nearest neighbors. It represents a balance between the per-token uncertainty and the expected benefit of crowdsourced answer to its neighbors. We compare both entropy metrics in the experiments of this paper.

5.2.2 Clustering Similar Tokens

Individual documents, especially citation data, may contain significant overlap between authors, conferences, publishers, etc. that appear in more than one document. While we accept a certain redundancy for quality control purposes, this should be tightly controlled through the AMT interface. Selecting the same tokens with the same labels from different documents essentially doubles the cost for a single answer.

Our solution to this problem is through a novel clustering technique. Tokens that share similar text and machine labeling properties have a high probability of sharing the same true labels. By mapping multiple tokens into the same *Question Cluster*, a single answer can be used to modify labels for a large number of tokens. Whereas before we constrained the token space to the set of documents, here we constrain the space even further to only the set of unique clusters. In the rest of this section we describe three specific cluster sets of cluster properties.

All cluster sets require tokens belonging to the cluster to share the same token text and machine labeling, but are set apart by different neighborhood constraints. Context is important because it's possible that two tokens with the same text such as the word "computer" could actually require different labelings and it would be false to cluster them together. The different cluster properties trade off accuracy and cluster size and we compare them in the experiments section. Figure **??** compares two similar tokens and illustrates the different cluster properties that are checked.

Same Field: The strictest of the cluster properties. It promotes the least amount of clustering, but contains greatest accuracy. The initial cluster representative defines a field by its CRF max likelihood label and all preceeding and suceeding tokens with the same label. Tokens are only added to the cluster if they share the entire field.

Same Label Neighbhorhood: A relaxing of the Same Field properties to only compare labels one position before and after the token. The existence of sub-entities such as a city that appears in more than one Proceedings field or an author that appears with different groups of authors motivate this approach of not checking the entire field. Clustering is greatly increased, but at the expense of a small amount of misclassification.

Same Token Neighborhood: The simplest of the cluster properties. We don't check any of the machine labelings, but measure redundancy based purely on the text associated with each token. Tokens are clustered together that share the same token as well as the same token directly preceeding and suceeding it. This has the advantage of being purely dependent on the data and not at all on the CRF output.

One paragraph about implementation of pseudo-code as database procedures somewhere.–SLG

5.3 Ordering Questions

The result of running one of the above clustering algorithms reduces the selection problem into one of selecting the *cluster* which provides the maximum reduction of uncertainty in the database. We consider three different ways of ranking clusters for top-k selection. Adhering to our information theoretic framework, it's possible to select the "representative" high entropy token used to initiate each cluster and rank by that token's individual entropy. On the other hand, if clusters are largely skewed in size, it may be more beneficial to rank by the actual size of the cluster. As a final heuristic attempting to combine both the entropy and cluster size approaches, we can sort by the total entropy of each cluster, that is, the sum of entropies of every token in the cluster. Our experiments compare and contrast these different ordering techniques.

6 Integration

Add paragraph on "frequentist" (MV) versus "Bayesian" approach–SLG One of the difficulties in relying on information from a crowd of sources is the possibility of a high degree of noise due to unreliable and in some cases even malicious sources. One of the standard procedures for increasing quality control is to increase the redundancy of questions. By asking the same question to multiple sources and aggregating the answers, we can achieve a higher probability of a good answer.

In many cases, it suffices to collect, say, 3 or 5 votes on each question and use the majority opinion. There are potential scenarios in which this ceases to be an effective strategy. If the probability of receiving low quality work is equal to or greater than that of receiving higher quality, it's detrimental to treat every vote of equal merit. Confusing or difficult questions can also cause conflict among the workers and result in a mix of answers. Taking the deterministic mode results in a loss of information about the controversy of the question, information which may prove useful in applications such as sentiment analysis or opinion-dominated questions.

Thus we are led to a desire to manifest the crowd response probabilistically, weighing votes proportionately and making decisions when conflicted on a question. We implement two approaches for this data integration task, drawing separately from probability theory and belief theory. The first maintains a single probability function, establishing a prior based on the machine's labeling, and updating the posterior using Bayes's Rule. Alternatively, we combine the Turker response in the absence of the machine prior using Dempster-Shafer theory. Both methods require an identification of the level of quality of each individual Turker. We describe previous work that we've leveraged in the next section before outlining our two integration methods.Still may want to add Halpern and Fagin reference.–SLG

6.1 Evaluating Turker Quality

Amazon Mechanical Turk provides no working system for maintaining the quality and reliability of their workforce and it is generally up to the Requester to ascertain such values on their own. The simplest system, known as "honey potting", is to carefully intermix questions for which the answer is known in advance and judge Turker performance against the gold standard. While generally effective, it lacks robustness and is defeatable to smart enough Turkers that can recognize them over time. More sophisticated methods estimate quality an unsupervised manner by judging each Turker's level of agreement with the mean set of answers. Examples include Bayesian [1, 8] methods and an approach using majority vote and expectation maximization [4].

We focus on a modified version of latter, attributable to Dawid and Skene [2], for implementation into CASTLE. For each question the EM algorithm takes a set of answers a_1,...,a_N provided by N Turkers assumed to be drawn from a categorical distribution. Associated with each Turker is a latent "confusion matrix" π_{ij}^{k} that designates the probability the k^{th} Turker will provide label j when true answer is i. Our modification simplifies to a binary accuracy variable $\pi^(k)$, which represents probability they will correctly label a question with the true answer. The goal of Dawid and Skene's EM algorithm is to recover π^k in the presence of the answers a_1^m,...,a_k^m for a set of questions $m \in M$.

In order to obtain a sufficient number of answers to similar questions by, HITs are designed in higher cost blocks. The single task of supplying a label to a token is worth around \$0.01. HITs are packaged in groups of 10 questions at \$0.10 each. This ensures that if K Turkers answer the HIT, relative performance can be judged across all 10 questions.

The algorithm initializes each Turker's accuracy to 1. It takes a majority vote among the answers to each question to define an initial answer set. Based on this agreed upon answer set, each Turker's accuracy π^k is computed. Another majority vote weighted by π^k determines a possibly different answer set. The Turker accuracies are re-computed. This process continues until convergence in both the "true" answer set and the π^k accuracies.

Let us take a moment to define precisely how we interpret Turker quality in the context of results of the EM algorithm. While the Dawid & Skene approach ultimately is calculated as correct or incorrect accuracies from a set of questions, we assume a different characteristic behavior associated with this score. Instead of the quality being a measure of whether we believe the Turker is "correct" or not, we take quality to be a measure of *reliability*. The quality score models the probability the Turker knows the correct answer and selects accordingly, while the inverse is the probability of a *random guess* from the set of possible answers. The two approaches in the next section tackle the problem of combining responses once we have an estimate of the Turkers' quality or reliability.

6.2 Two Approaches to Integration

6.2.1 Bayesian Conditional Probability

The fundamental assumption taken with the Bayesian model is that the ML extracted values present a serviceable prior probability over the choice of labels. For a well-trained machine model, its output can be used as starting point upon which additional evidence from the crowd is used to adjust the label decision in the right direction. The machine acts as a regularizer, the more peaked any aspect of the original output distribution the more impact the prior plays and consequently the greater the trust placed in the original model.

Let A_1^n,...,A_K^n be a set of categorical random variables corresponding to the answers received from K Turkers for question n. The CRF's original output, a random variable L which also follows a categorical distribution over the label space, is our current estimate of the true distribution of labels fora specific token. The integration problem is to find the posterior $P(L^n|A_1^n,...,A_K^n)$ conditioned on the answers provided by the Turkers. This can be calculated using Bayes's Rule:

$$P(L^n|A_1^n, ..., A_K^n) = \frac{P(A_1^n, ..., A_K^n|L^n)P(L^n)}{P(A)} \tag{4}$$

Since the set of answers is fixed and we're only concerned with relative differences among different label possibilities, we may without loss of generality focus solely on the numerator. The initial prior, $P(L)$, is just the CRF's marginal probability before considering any new evidence. The evidence term, $P(A_1^n, ..., A_K^n|L^n)$, represents the probability the Turker answers were generated from a specific true label. Our Bayesian model assumes Turker quality is an adequate measure of their agreement with the true label,

$$P(A_1^n, ..., A_K^n|L^n) = \prod_k P(A_k^n|L^n) \tag{5}$$

$$P(A_k^n = a|L^n = l) = \mathbb{1}_{a=l} * Q_k + (1 - Q_k) * \frac{1}{|L|} \tag{6}$$

where a and l are values drawn from the label space and Q_k is the quality of the kth worker. Equation 5 follows from all Turker answers being independent of each other and equation 6 simply restates our assumption about the use of Turkery quality Q_k. If the answer matches the label l, the first term on the right hand side is the probability the Turker is reliable and answers the question truthfully. The second term incorporates the probability they are unreliable or a spammer and through *random guessing* finds the correct answer with probability $1/|L|$, $|L|$ being the number of possible labels. If they don't match, we have the probability the Turker is unreliable, $1 - Q$, and the probability a random guess produces an incorrect answer, $(L - 1)/L$.

The full model is

$$P(L^n = l|A_1^n = a_1, ..., A_K^n = a_k) = \\ P(L^n = l) \prod_k \left(\mathbb{1}_{a_k=l} * Q_k + (1 - Q_k) * \frac{1}{|L|}\right) \tag{7}$$

Using equation 7 for all possible labels l and renormalizing produces a new posterior distribution accounting for both the initial ML extracted result and evidence gathered from the crowd. The product can be extended and updated as new evidence comes in over time. While currently evidence is designed to come from the crowd in CASTLE, there is no explicit restriction preventing future updates from incorporating evidence from a number of different extractions as well as the crowd. We conclude this section with an explicit example.

Probably want to change these numbers. What differences between DS and Bayes do I want to exhibit by using specific numbers?– SLG EXAMPLE 1. *Assume a binary question is answered by two Turkers. Turker A has quality 0.8 and answers label 0, while Turker B has quality 0.6 and answers label 1. The prior CRF marginal probability over {0,1} is {0.3,0.7}. We want to ascertain the com-*

bined distribution for the label L. According to equation 7,

$$\begin{aligned} P(L=0|A,B) &= \frac{1}{Z}P(L=0)P(L=0|A)P(L=0|B) \\ &= (0.4)*(0.9)*(0.2)*\frac{1}{Z} \\ &= .072*\frac{1}{Z} \end{aligned} \quad (8)$$

$$\begin{aligned} P(L=1|A,B) &= \frac{1}{Z}P(L=1)P(L=1|A)P(L=1|B) \\ &= (0.6)*(0.1)*(0.8)*\frac{1}{Z} \\ &= .048*\frac{1}{Z} \end{aligned} \quad (9)$$

After combining and normalizing, the final distribution over {0,1} is {0.6,0.4}. While the CRF originally favored label 1, the new distribution favors label 0.

6.2.2 Dempster-Shafer Evidential Combination

Without explicit reference to the CRF prior, we're left with the task of *combining* disparate evidence from a group of Turkers. This can be accomplished using Dempster's Rule of Combination, which operates over a set of mass functions. Mass functions differ from probability functions by relaxing the Kolmogorov axiom that functions must sum to 1.

7 Experiments

In this section we demonstrate the effectiveness of our selection and integration approaches on sets of both synthetic and real data. We extracted 14,000 labeled citations from DBLP footnote 1–SLG and 500,000 from the PubMed database footnote 2–SLG. For unlabeled testing data, we removed the labels and concatenated text from each of the available fields. Order of fields was occasionally mixed in keeping with real-life inconsistency of citation structure.

7.1 Experiments w/ Synthetic Data

7.1.1 Selection

Figures 3, 4, and 5 contain experiments comparing our various selection algorithms by detailing the accuracy improvements for each question asked. Tokens were selected using a specific combination of seeding, clustering, and ranking approaches.

Initially, a token was selected from each document using some seeding mechanism. The number of questions is prohibitively large to show the full range of our methods, so we automatically answer each question with its ground truth label. It's shown in the next section that the high accuracy of Mechanical Turk answers allow this to be a working assumption. The same answer (label) to the question (token) is applied to all subsequent tokens in its cluster. A constrained Viterbi inference algorithm runs over all documents containing tokens belonging to question clusters. The accuracy value in each figure represents the final token accuracy after running constrained inference.

In this paper, we proposed two possible functions for selecting a token from each document. High Entropy chooses that which has the highest marginal entropy over its labels while Neighborhood Entropy selects the token in the center of the largest 3-window pocket of marginal entropies. Figure 3 shows effectiveness of both methods when compared to randomly selecting a token for both High Entropy and Total Entropy ranking. The default clustering is Same Label Neighborhood. In both cases, Neighborhood Entropy maintains a consistently higher accuracy, lending evidence to the idea that constrained inference has a larger effect on pockets of high entropy than it does on the single highest entropy tokens.

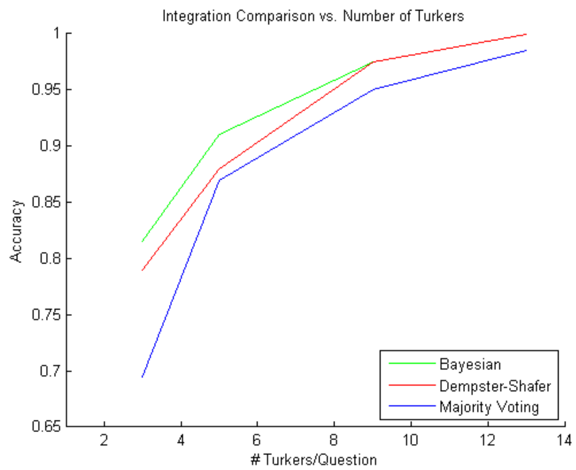


Figure 6: Comparison of integration methods vs. number of Turkers per question.

Both methods double the overall possible accuracy improvement with fewer questions. For some accuracy regions even orders of magnitude fewer questions are needed.

Figure 4 compares the possible clustering algorithms for the High Entropy and Total Entropy ranking functions. All use high entropy for seeding. Clustering by similar tokens that have the same label and share preceding and succeeding labels produce the largest clusters with the greatest net effect. For the DBLP set, there were zero clustering errors for Same Token and Same Field, and approximately 2% of citations were clustered incorrectly using the Same Label approach. As the figures prove, however, the benefit of larger clusters far outweigh the additional errors.

The final set of synthetic selection experiments is shown in Figure /reffig:selection3. While it initially seemed like a heuristic, the effectiveness of Total entropy for ranking should now be apparent. For both high entropy and random seeding, total entropy combines the early question strength of large clusters and the late question power of high entropy. Same Label Neighborhood is again the default clustering for all ranking comparisons. It's important to note, that even for random seeding, Total Entropy outperforms everything else.

7.1.2 Integration

Describe process for producing synthetic workload and justify simplifying assumptions.–SLG

Exp4: DS vs. MV vs. Bayes for varying number of Turkers.–SLG

Exp5: DS vs. MV vs. Bayes for varying levels of mean Turker quality.–SLG

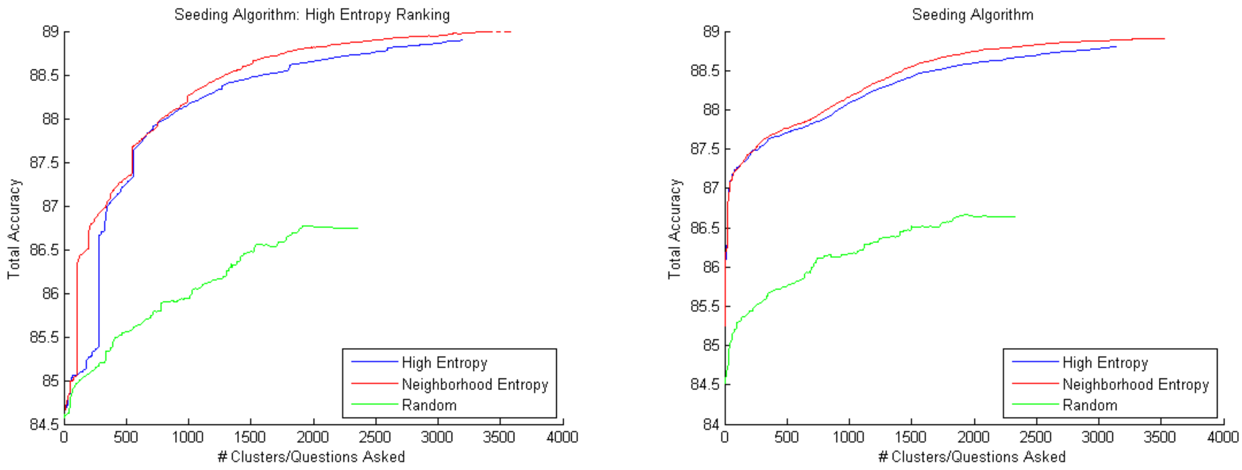
Exp6: Recall vs. accuracy for varying entropy thresholds.–SLG

7.2 Experiments w/ Real Data

Description of real experiment methodology.–SLG

Exp7: Table of accuracy comparisons for DS, MV, and Bayes before and after edits plus clamped inference for both data sets.–SLG

Exp8: Recall vs. accuracy for varying entropy thresholds for both data sets.–SLG



(a) High Entropy

(b) Total Entropy

Figure 3: Seeding comparison for high entropy and total entropy ranking.

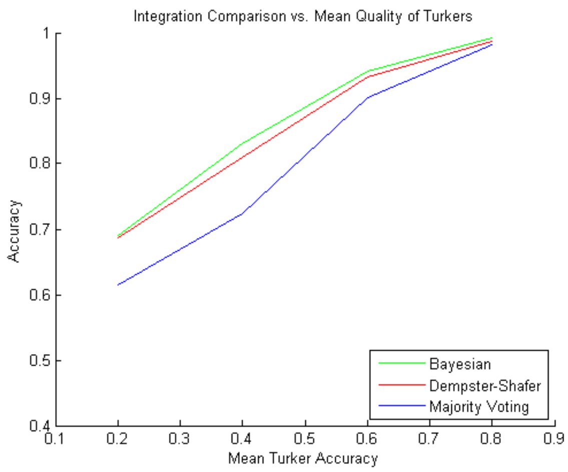


Figure 7: Comparison of integration methods vs. average Turker quality.

8 Conclusion

9 Future Work

10 References

[1] B. Carpenter. Multilevel Bayesian Models of Categorical Data Annotation. Technical report, Alias-i, 2008.

[2] A. P. Dawid and A. M. Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):pp. 20–28, 1979.

[3] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In Sellis et al. [9], pages 61–72.

[4] P. G. Ipeirotis, F. Provost, and J. Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '10, pages 64–67, New York, NY, USA, 2010. ACM.

[5] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.

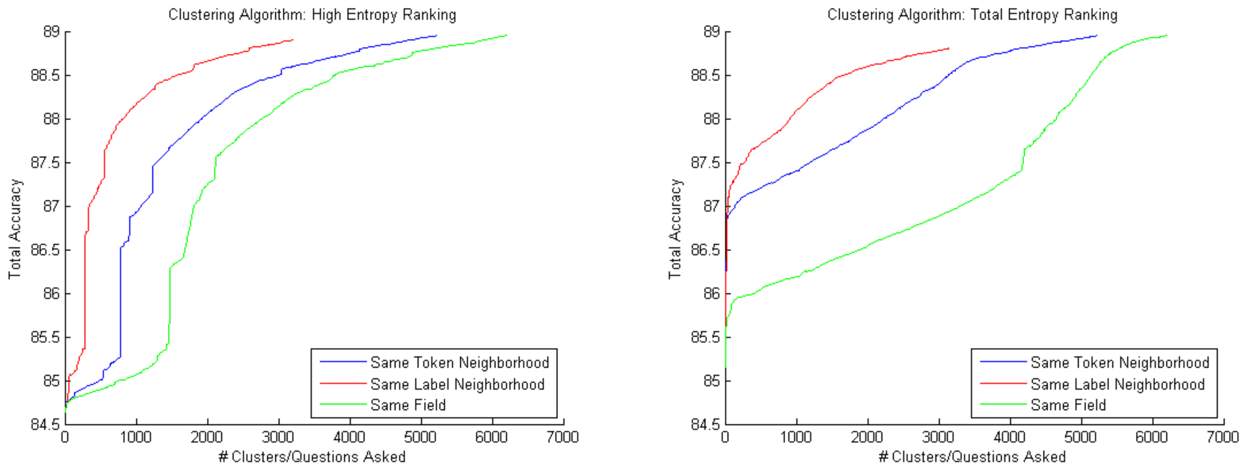
[6] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Demonstration of qurk: a query processor for humanoperators. In Sellis et al. [9], pages 1315–1318.

[7] E. J. Quinn, B. B. Bederson, T. Yeh, and J. Lin. Crowdflow: Integrating machine learning with mechanical turk for speed-cost-quality flexibility. Technical report, 2010.

[8] V. C. Raykar and S. Yu. Eliminating spammers and ranking annotators for crowdsourced labeling tasks. *Journal of Machine Learning Research*, 13:491–518, 2012.

[9] T. K. Sellis, R. J. Miller, A. Kementsietsidis, and Y. Velegrakis, editors. *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*. ACM, 2011.

[10] D. Z. Wang, M. J. Franklin, M. N. Garofalakis, and J. M. Hellerstein. Querying probabilistic information extraction.



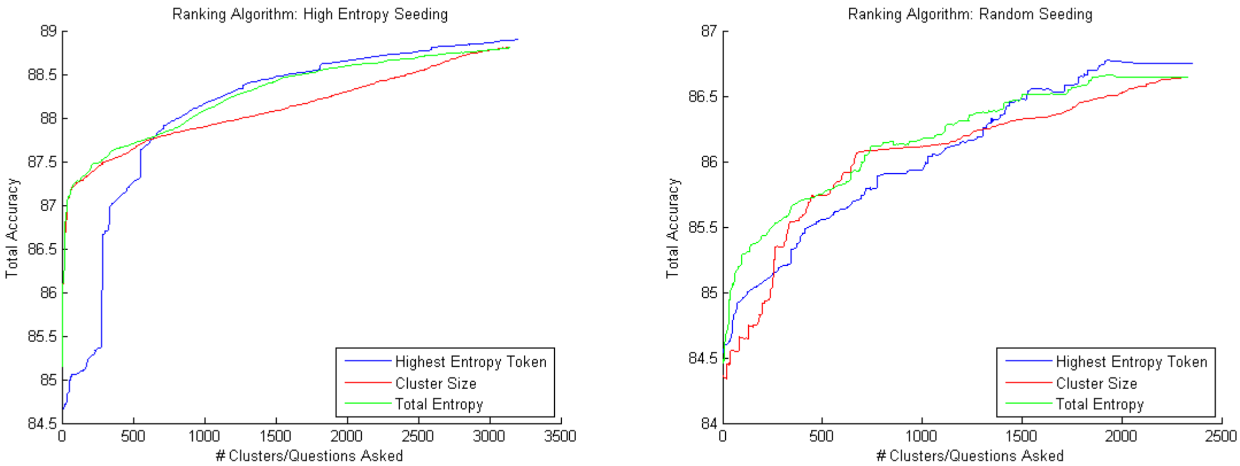
(a) High Entropy

(b) Total Entropy

Figure 4: Clustering comparison for high entropy and total entropy ranking.

PVLDB, 3(1):1057–1067, 2010.

11 Appendix



(a) High Entropy

(b) Random

Figure 5: Ranking comparison for high entropy and total entropy ranking.