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 [6] 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.[9], 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 [9]. 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

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.

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

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

Add some reference to Guestrin's paper and selection over dependent vs. independent examples.–SLG CASTLE is unique from other crowdsourced databases in that it doesn't require fields be empty to crowdsource the value. Its strength lies in its ability to analyze the probabilistic distributions of over its fields and determine where the largest benefit from human feedback would occur. Our approach is similar to uncertainty sampling, where high entropy examples are most informative to the model and are chosen to be labeled and added to the training set. We make a slightly different assumption. Here, we map entropy inversely into confidence. The more spiked a distribution over a single label, corresponding to low entropy, the greater the confidence in the result. More uniformity is characteristic of higher confusion and more likely to lead to an incorrect result. The focus is thus on ordering tokens by greatest entropy so the top-k can be submitted to the crowd given a set budget.

5.1 Calculating Entropy

There are many subprocesses which go into the calculation of token entropy. The first is creation of two new tables for the Forward and Backward values, colloquially known as the α and β values. In linear-chain CRFs, $\alpha(j, s)$ represents the sum of scores of all paths from position 1 to position j that pass through label s. It's calculated by the recurrence:

$$\alpha(0, y_i) = \frac{1}{|L|} \tag{1}$$

$$\alpha(j, y_i) = \sum_{y_{i-1}} [\sum_k \lambda_k f_k(y_i, y_{i-1}, x_i) + \alpha(j-1, y_{i-1})] \tag{2}$$

where $|L|$ denotes the size of the label space. Initialization simply makes all transitions into the first set of labels equally likely. The function produceForwardTable(docID) uses the factor table and token table to populate FORWARDTBL, represented as a matrix of size $|N| \times |L|$ based on the recurrence in (2).

The set of Backward values are calculated with a similar recurrence:

$$\beta(N, y_i) = 1 \tag{3}$$

$$\beta(j, y_i) = \sum_{y_{i-1}} [\sum_k \lambda_k f_k(y_i, y_{i-1}, x_i) + \beta(j+1, y_{i-1})] \tag{4}$$

The β values represent the sum of scores of all paths from position i to position N. BACKWARDTBL is the same size as FORWARDTBL and similarly computed from its own recurrence upon calling produceBackwardTable(docID).

The marginal distributions are represented as an element-by-element product of FORWARDTBL and BACKWARDTBL. Explicitly, for each element of MARGINALTBL:

$$M(j, y_i) = \alpha(j, y_i) \times \beta(j, y_i) \tag{5}$$

Normalizing over the columns gives a marginal distribution for each position j. Finally, entropy is calculated and stored in the relational ENTROPYTBL.

ENTROPYTBL(docID, pos, token, entropy)

Though computed over a probabilistic distribution, entropy is a deterministic quantity in the database. It's values at position j are calculated in the usual way:

$$E(j) = -\sum_{y_i} M(j, y_i) log(M(j, y_i)) \tag{6}$$

5.2 Optimizations

5.2.1 Seeding

The probabilistic component of each token tuple, label^p, is determined by the feature functions in FACTORTBL through the Viterbi algorithm. Far from being independent, field values are deeply connected between tuples and modifying one label field in a document d has wide-ranging implications for the other probabilistic fields, most importantly for those tokens preceeding and suceeding it.

For example, consider the incorrectly labeled document in Figure **??**. In cleaning up this document at the token level, it appears we would need to ask a question about both tokens "X" and "Y" and have them changed from a Title label to an Author label. Running the constrained Viterbi inference algorithm provides additional power to be harnessed. Say just the token "X" were queried and submitted to the crowd and the label received back was an Author. The CRF has learned that sequences such as Title-Author-Title do not appear in the corpus and inference constraining "X" to

be an Author would also select "Y" as Author. This is an example of killing two birds with one stone and is just a simple example. Crowd answers that differ more drastically from the original inference result will experience even greater changes in the constrained inference, further enhancing the power of just a single question.

This strength coming from constrained inference leads to a possible weakness in the question selection process. Selecting both tokens "X" and "Y" leads to twice the cost compared to just asking one of them. Figure **??** shows the entropy distribution over tokens for a typical text-string in CASTLE. Entropy typically peaks to high values at the boundaries where labels change and multiple high values appear in the same neighborhood. To reduce redundancy in asking multiple tokens at nearly the same position, we limit each batch of questions to include at most one token from each document.

Thus we have a one-to-one mapping between documents d and the highest entropy token $\hat{t}$. From here on own we use the phrases "querying a document" and "querying a token" interchangeably, there the token in question always refers to the highest entropy token $\hat{t} \in \mathcal{T}$ associated with d

5.2.2 Clustering

In our data model, tokens represent the underlying primitives of the database. Previously, we described calculation of a token's label entropy for the purpose of discovering the tokens most in need of human correction or confirmation. Selection based purely on entropy, however, leads to less than optimal results in practice.

In information extraction, and particularly citation extraction, important entities show up again and again. Repeated authors, conferences, publishers, etc. lead to a certain level of redundancy among individual tokens. It would be inefficient for the crowd to label the same token multiple times from different questions, leading to higher costs and longer wait times for relatively minimal improvement.

In CASTLE we eliminate this problem with a novel clustering scheme. We say $d \in \mathcal{C}$, for some cluster $\mathcal{C}$ if the following conditions are satisfied $\forall d_i \in \mathcal{C}$:

1. d.token = d_i.token
2. d.prevLabel = d_i.prevLabel
3. d.label = d_i.label
4. d.nextLabel = d_i.nextLabel

Thus clusters of documents whose high entropy tokens are equivalent and share similar contexts are grouped together. Answers retrieved for one document are applied to all documents in the same cluster. Utilizing context is important because the same token may appear in more than one field location, such as a Title in one and a Conference in another. It's also common for Conferences and Proceedings to share tokens within the same citation. This is the basis behind requiring labels for all tokens in the cluster to be the same. The inclusion of previous and succeeding labels lowers the probability of a mislabeling leading to a false clustering, ensuring that the relative positions of all tokens in a cluster are the same.

5.2.3 Ranking

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, one achieves a higher probability of a high quality answer.

In many cases, it suffices to collect 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 is detrimental to treat every vote of equal merit. Confusing or difficult questions may 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 opinionated questions.

Thus we are led to a desire to manifest the crowd response probabilistically, weighting votes proportionately and making decisions when it is conflicted on a question. We develop a principled framework rooted in belief theory for combining crowd data. There are two fundamentally different ways of interpreting belief theory [3], that of a generalized probability distribution (Bayesian) and as a means to combine different forms of evidence (Dempster-Shafer). We show these to be equivalent if crowd answers are mapped to a certain distribution based on the quality of each worker. We describe our method for uncovering this quality in the next section.

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, 7] 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. The full pseudo-code can be found in the appendix.

6.2 Two Views of Belief Theory for Aggregation

The reason for submitting to belief theory as our main tool in the aggregation of the Turkers and machine is that it provides a nat-

ural framework arriving at a posterior distribution composed of various pieces of evidence. While the roots of belief theory first centered around the Dempster-Shafer model, much criticism has been laid upon the model for turning up erroneous or inaccurate results. Halpern and Fagin [3] argue this is purely from a misuse of appropriating one interpretation for another. The first view of belief function one can take is that of a generalized probability function, starting with a prior probability and updating as new evidence comes along to arrive at a conditional posterior. On the other hand, viewing belief functions as evidence themselves leads one to use Dempster's Rule of Combination. One presents the *updating* of evidence while they other presents the *combining*. One utilizes a prior while the other does not.

We use this as inspiration for studying two different approaches to aggregating humans and machines akin to the differing interpretations. In our Bayesian formulation, the CRF marginal distribution is used as a prior and *updated* based on Turker responses. Using an alternative Dempster-Shafer model, we forego the use of a prior and *combine* Turker responses using Dempster's Rule of Combination.

6.2.1 Bayesian Conditional Probability

The fundamental assumption taken with the Bayesian model is that the ML extracted values present a serviceable prior to the model. For a well-trained model, this view is that the machine is *close enough* and the crowd's response is presented as a tweak, pushing its decision in one direction or another. One interpretation is of the machine as a regularizer and the more peaked any aspect of the machine distribution is the more impact the prior plays and consequently the greater trust is placed in the original model.

Let A_1^n,...,A_K^n be a set of categorical random variables corresponding to answers received from K Turkers for question n. The CRF's prior, a random variable L which also follows a categorical distribution over the label space, is our current estimate of the true distribution of labels for the token in question. The aggregation 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 found using Bayes's Rule:

$$P(L^n|A_1^n,...,A_K^n) = \frac{P(A_1^n,...,A_K^n|L^n)P(L^n)}{P(A)} \tag{7}$$

We make the simplifying assumption without loss of generality that marginal probability of any set of answers $P(A)$ is uniform and only focus on the numerator. It's already been stated that $P(L)$ is just the CRF's marginal probability before considering any new evidence. The evidence term can be modeled with the same assumption that was made in using Dawid and Skene to estimate Turker quality.

$$P(A_1^n,...,A_K^n|L^n) = \prod_k P(A_k^n|L^n) \tag{8}$$

$$P(A_k^n = a|L^n = l) = |\mathbb{1}_{a \neq l} - Q_k| \tag{9}$$

where a and l are values drawn from the label space and Q_k is the quality of the kth worker. Equation 8 follows from all Turker answers being independent of each other and equation 9 simply restates our assumption about the use of Turkery quality Q_k. The probability of a Turker's answer a agreeing with the true label l equals Q_k, while the probability of an incorrect answer where they disagree is $1 - Q_k$. The full model is

$$P(L^n = l|A_1^n = a_1,...,A_K^n = a_k) = P(L^n = l)\prod_k |\mathbb{1}_{a_k \neq l} - Q_k| \tag{10}$$

Using equation 10 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.

6.2.2 Dempster-Shafer Evidential Combination

A viable alternative is to exhibit no faith in the machine's initial marginal calculation. After all, one could argue that by selecting only the most uncertain tokens that metric loses its value. Tossing out 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.

While the Bayesian approach was inspired by an alternative interpretation of belief functions, the actual implementation is still a probability function through and through, with all of Kolmogorov's axioms defining a probability function still holding. For evidential combination, however, we leverage the full power of belief theory and relax some of those axioms to map to a set of belief functions. The main difference between a belief function and a probability function is that probability functions are defined only over the *measurable* subsets of a set while belief functions are defined over *all* subsets (the power set) of a set [8].

Mapping the data from Turkers into belief or mass functions is relatively straightforward. Like with the Bayesian approach, our confidence in them getting the answer correct is reflected in their Quality score. The mass function $m(a_k)$ gets assigned the score Q_k. Let $\mathcal{A}$ be the set of all possible labels $1, 2, ..., L$. Intuitively, $m(\mathcal{A})$ is the mass associated with a random guess and all L labels being equally likely. We assume in this framework that Turkers are either reliable, getting the answer correct with belief score Q_k, or unreliable, reflected in a random guess with belief score $1 - Q_k$. Explicitly, for a Turker k with provided answer a_k:

$$m^n(2^L) = 0 m^n(a_k) = Q_k m^n(\mathcal{A}) = 1 - Q_k \tag{11}$$

The first equation simply states that initialize all mass functions to zero before setting the two values below. The mass function $m(\mathcal{A}$ has no meaning in standard probability theory, as the set of all outcomes is not a measurable in the probabilistic sense. We use it mainly as bookkeeping for the uncertainty in the result before normalizing it out when the aggregation computation is completed. The set of mass functions from multiple Turkers can be combined using Dempster's Rule of Combination between Turker 1 and Turker 2 for each set $A \in 2^L$:

$$\begin{aligned} m_{0,1}(A) &= (m_1 \oplus m_2)(A) \\ &= \frac{1}{1-K} \sum_{B \cap C = A \neq \emptyset} m_1(B) m_2(C) \end{aligned} \tag{12}$$

$$K = \sum_{B \bigcap C = \emptyset} m_1(B) m_2(C) \tag{13}$$

The procedure is to map all HIT responses to mass functions and combine them one-by-one in turn to produce a single combined

mass function. Any remaining uncertainty in $m_{comb}(\mathcal{A})$ is added to all the singleton functions and re-normalized to produce a single probability function. The original belief formulation is maintained in CASTLE for easy combination if new evidence arrives at a later time.

While we introduce Dempster-Shafer theory in the context of our simpler one-answer-per-question framework currently found in CASTLE, the method will become more powerful in future work when we plan to extend functionality to allow the Turkers to provide more than one response per question when uncertain. Reasoning over such fuzzy sets exemplifies the real power for using belief theory over probability theory.

6.3 Thresholding the Turkers

Even human computation is not perfect. The previous section looked at ways to combine Turker answers probabilistically to arrive at a final result that is not deterministic. This is useful for when there is controversy or confusion elicited over the answers of a question. Placing a cap on the entropy of the final probability function is a good way to guarantee only high quality returns on crowdsourced answers. Answer aggregations not falling under the cap have their answers retained and questions are re-submitted during the next batch to gain further evidence in reaching a decision.

6.4 Probabilistic Integration

Not only does CASTLE have the ability to aggregate answers from multiple sources, but also the ability to reinsert the resulting distribution back into the CRF. Since the underlying architecture of the system is a CRF, the dependence properties of each field are made explicit and re-running the inference algorithm has the potential to change surrounding fields as well. This "constrained inference" substitutes the aggregated marginal distribution of a token in for the computed transition probabilities. This highlights a very strong advantage of CASTLE system, in that large errors can be corrected by small, incremental changes.

7 Experiments

Introduction to various experiments. Description of data sets.–SLG
In this section we demonstrate the efficacy 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. The data originally came in XML and JSON formats respectively. For unlabeled testing data, we extracted 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 Synthetic Experiments

7.1.1 Selection

Exp1: Comparison of clustering algorithms.–SLG
Exp2: Comparison of ranking/seeding functions.–SLG
Exp3: Accuracy before and after introduction of crowd edits and clamped inference–SLG

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 Real Experiments

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

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] J. Y. Halpern and R. Fagin. Two views of belief: Belief as generalized probability and belief as evidence. *Artif. Intell.*, 54(2):275–317, 1992.

[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] 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.

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

[8] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, 1976.

[9] D. Z. Wang, M. J. Franklin, M. N. Garofalakis, and J. M. Hellerstein. Querying probabilistic information extraction. *PVLDB*, 3(1):1057–1067, 2010.

11 Appendix