Crowdsourcing Algorithms: Summary and Comparisons

Daniel Khashabi, Stephen Mayhew

May 5, 2015

1 Introduction

Given a specific setting for the crowdsourcing task[1], we implemented many methods, and attempted a few approaches which are of slight interest.

2 Methods

2.1 Majority Voting and Power Iteration

The two simplest baselines are majority voting (stylized as `mv` in our code) and the singular value approach, which we call `power_iteration`. This latter approach is to simply take the leading left singular eigenvector of the matrix A. We found that we needed to do some fiddling with the eigenvector because it would randomly be flipped all negative.

2.2 Belief Propagation and variants

Following from **?**, we begin with the following BP updates:

$$y^{(k)}_{a\to i}(p_a) \propto \mathcal{F}(p_a) \prod_{j\in\partial a\setminus i} \Big\{ (p_a + \bar{p}_a + (p_a - \bar{p}_a)A_{ja}) x^{(k)}_{j\to a}(+1) + (p_a + \bar{p}_a - (p_a - \bar{p}_a)A_{ja}) x^{(k)}_{j\to a}(-1) \Big\}$$

$$x^{(k+1)}_{i\to a}(\hat{t}_i) \propto \prod_{b\in\partial i\setminus a} \int \Big(y^{(k)}_{b\to i}(p_b)(p_b \mathbb{I}_{(A_{ib}=\hat{t}_i)} + \bar{p}_b \mathbb{I}_{(A_{ib}\neq\hat{t}_i)}) \Big) dp_b$$

With the Haldane prior, these updates collapse into a simple iterative algorithm, as seen in **?**. In our implementation, we refer to this algorithm as `simplified_bp`. We found this was quite slow, although it may have been poor implementation.

However, the Haldane prior is much too simple, and we would like to have a way to incorporate a better guess at the priors (especially when we know that the prior is a Beta distribution). But

[1]`http://web.engr.illinois.edu/ swoh/courses/IE598/hw/project2015.pdf`

there is a problem with implementing the above updates directly: the continuous p_j's. A simple solution is to discretize each p_j into several chunks, and thereby crudely approximate the integral. We did this, and the second update became:

$$x_{i\to a}^{(k+1)}(\hat{t}_i) \propto \prod_{b\in\partial i\backslash a} \sum_{p_b\in P} \left(y_{b\to i}^{(k)}(p_b)(p_b \mathbb{I}_{(A_{ib}=\hat{t}_i)} + \bar{p}_b \mathbb{I}_{(A_{ib}\neq\hat{t}_i)}) \right)$$

where $P = \{p_1, p_2, ..., p_k\}$. We found that this was very slow, although this may be due to a poor implementation. In our code, we refer to the implementation as `discretized_bp`.

2.3 Expectation-Maximization

We implemented the EM derivation found in the slides[2]:

E-Step:

$$q_i(t_i) = \mu_p(t_i \mid A)$$

M-Step:

$$p_j = \frac{\sum_{i\in\partial j} q_i(A_{ij})}{|\partial j|}$$

This implementation is called `em` in our code.

Further, we derived a version of EM that included Beta priors (see the appendix for full derivation).

$$p_j = \frac{\alpha_j - 1 + \sum_j q(A_{ij})}{\alpha_j + \beta_j - 2 + \sum_j q(A_{ij}) + \sum_j q(-A_{ij})}$$

This implementation is called `em_with_priors` in our code. (In implementation, we used $\alpha = 6$ and $\beta = 2$, because this is how the data was created. This seems like cheating, but we think of it as though we made an exceptionally good guess).

2.4 HITS

Hyperlink-Induced Topic Search (HITS), also known as hubs and authorities, is a simple web ranking algorithm that is often used in trustworthiness applications, which are very similar to crowdsourcing problems. A full derivation is given in **?**, but by way of completeness, we give a quick overview here.

Given a graph, the algorithm finds a hub score and an authority score for each node. Originally the nodes were webpages and the graph had no particular pattern; in this case, we have worker nodes and task nodes, and the graph is bipartite.

We initialize y in some suitable fashion, and then iterate:

$$x^{(p)} = \sum_{q:(q,p)\in E} y^{(q)}$$

[2] `http://web.engr.illinois.edu/ swoh/courses/IE598/handout/crowd.pdf`

$$y^{(p)} = \sum_{q:(p,q)\in E} x^{(q)}$$

normalizing the x and y vectors at each iteration.

Although HITS is very similar to simplified BP, it is important to note that HITS operates on a different graph. Where simplified BP produces results close to the leading singular eigenvector at of the adjacency matrix, $A \in \{0, -1, 1\}^{(m\times n)}$, HITS produces results equal to the principal eigenvector of the adjacency matrix $A \in \{0, 1\}^{(m\times n)}$.

Ini our implementation of HITS, since we generate the graph in the form used by simplified BP (that is, with negative values), we need to perform a simple rewrite. With matrix dimensions given as $m \times n$, we expand it to $m \times 2n$, such that each column becomes two columns. Each -1 becomes $[1 \;\; 0]$ and each 1 becomes $[0 \;\; 1]$.

In our code, this algorithm is referred to as `hits`.

2.5 Iterative Weighted Majority Voting

We also implemented the algorithm found in **?**, which is basically hard EM, with an added step which rescales the estimated worker weights w_i:

$$\nu_i = Lw_i - 1$$

In this case, $L = 2$, which is reminiscent of the q variable measuring quality in **?**. The reasoning given is also similar. Estimated weights close to 0.5 (that is, close to random) will be pushed to 0, and higher weights will be rewarded. ν_i are then used as the weights for weighted majority voting.

We refer to this algorithm as `iwmv` in our code.

3 Experiments

We implemented these algorithms in matlab. We ran `mv`, `em`, `em_with_priors`, `power_iteration`, `hits`, `iwmv` together with the full parameters specified in the problem statement. That is, $n = 2000$ averaged over 100 instances. See Figure **??** for a plot of these algorithms.

`simplified_bp` and `discretized_bp` were too slow to run on the full parameter set, so we ran them on smaller parameter sizes to get an estimate for the scores. See Figure **??** for `simplified_bp` and Figure **??** for `discretized_bp`.

For the discretized BP case, as seen in Figure **??**, we discretized the interval $[0, 1]$ with 10 chunks and some slight rounding to avoid 0 and 1 (which have 0 probabilities in the Beta distribution). The final error probability is 0.006346, which is slightly better than the best result of all the other algorithms on the much larger problem size.

We were surprised that `iwmv` should work so well.

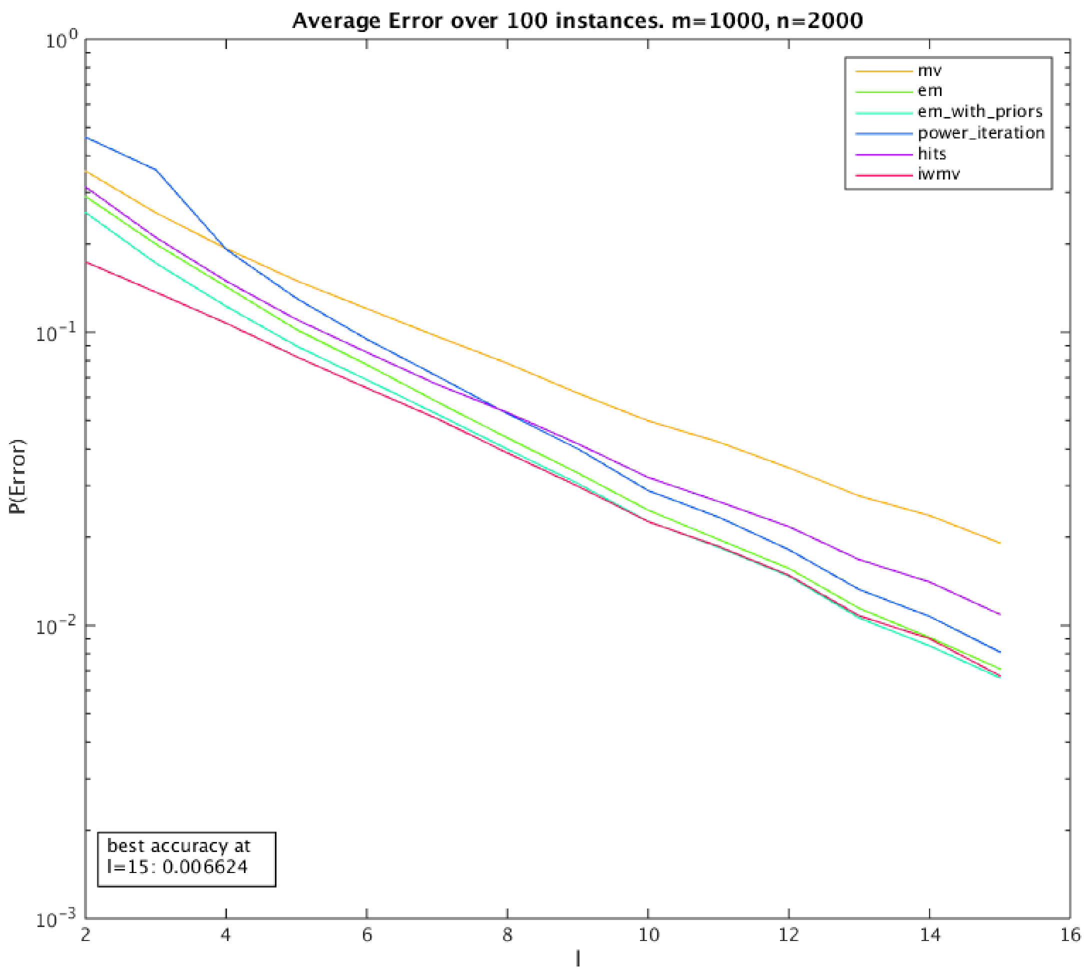

Figure 1: Many runs at high parameter values.

All of our code is available on GitHub[3].

4 Further Work

We put considerable time into implementations of tree-reweighted message passing (see **?**), and spectral initialization for EM (see **?**). However, neither of them came to fruition as of the time of writing (perhaps we can work out the bugs before the presentation in 12 hours). We were attempting to use opengm[4].

[3]`https://github.com/mayhewsw/crowdsourcing/`

[4]`http://hci.iwr.uni-heidelberg.de/opengm2/`

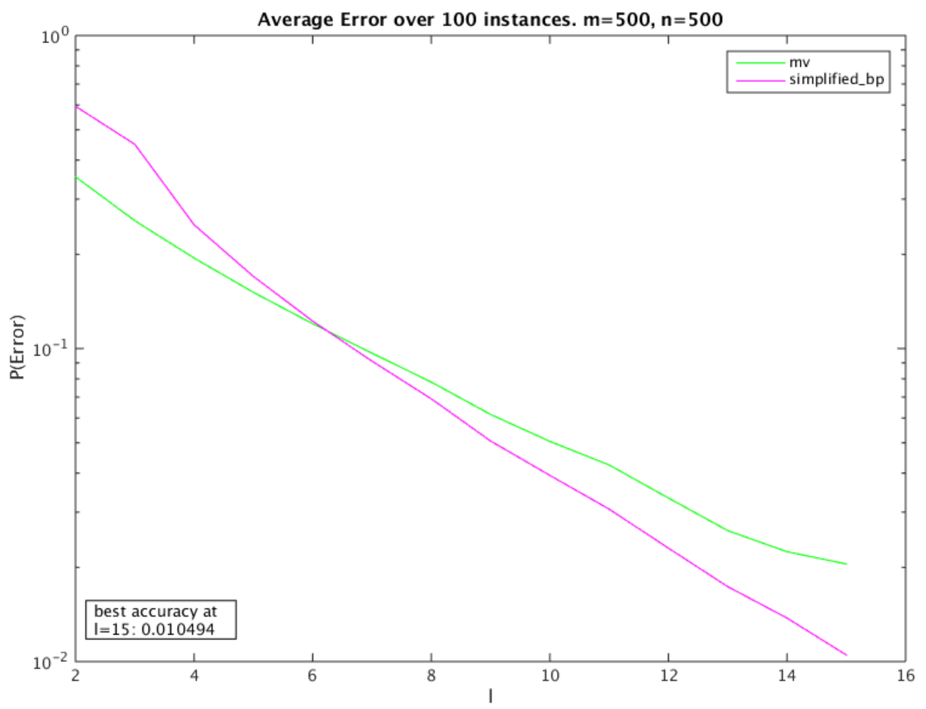

Figure 2: Simplified BP with manageable parameters. (Missing... as of writing, this graphic is not available because of computer issues)

References

Karger, D. R., Oh, S., and Shah, D. (2014). Budget-optimal task allocation for reliable crowdsourcing systems. *Operations Research*, 62(1):1–24.

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632.

Kolmogorov, V. (2006). Convergent tree-reweighted message passing for energy minimization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(10):1568–1583.

Li, H. and Yu, B. (2014). Error rate bounds and iterative weighted majority voting for crowdsourcing. *arXiv preprint arXiv:1411.4086*.

Zhang, Y., Chen, X., Zhou, D., and Jordan, M. I. (2014). Spectral methods meet em: A provably optimal algorithm for crowdsourcing. In *Advances in Neural Information Processing Systems*, pages 1260–1268.

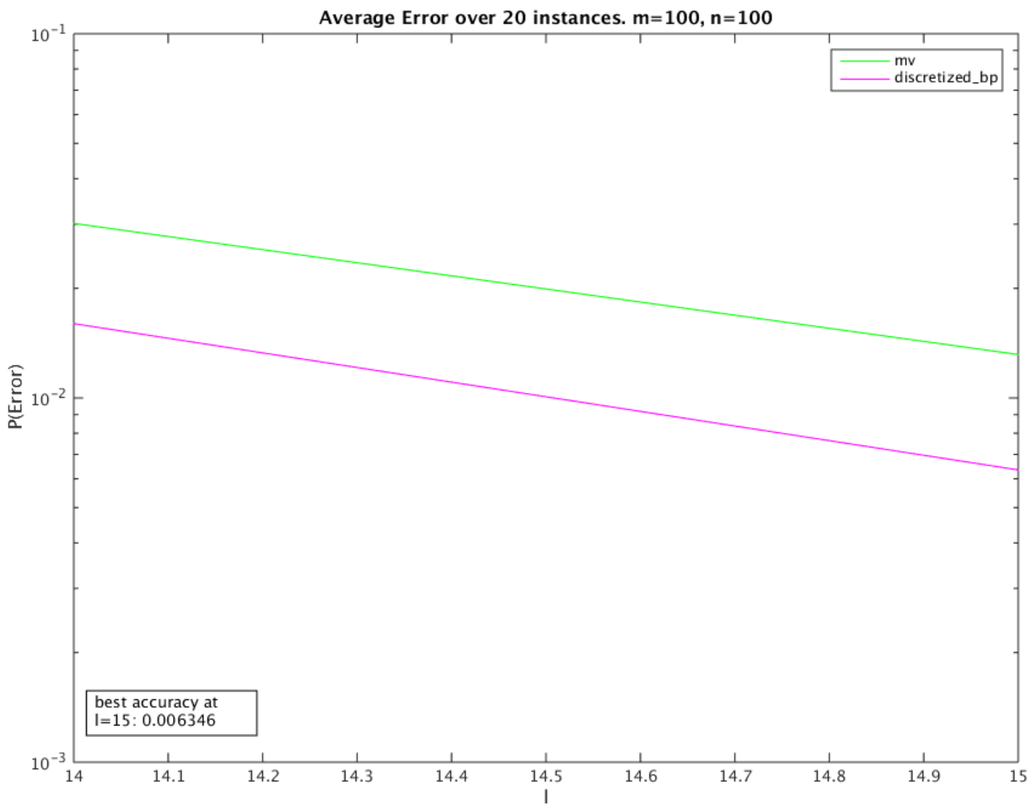

Figure 3: Majority Voting and Discretized BP. (This is over a small problem size, and only for $\ell = 14$ and $\ell = 15$ because of slow running times)

5 Appendix

EM derivation

Define

$$g_{ij}(t_i, p_j) = p_j \mathbf{I}\{A_{ij} = t_i\} + (1 - p_j)\mathbf{I}\{A_{ij} \neq t_i\}$$

Consider having the following prior on each p_j:

$$c + (1-c)Z_j, \quad \text{where } Z_j \sim \text{Beta}(\alpha_j, \beta_j)$$

The full likelihood: $L(p;t) = \prod_{i,j} g_{ij}(t_i, p_j) \prod_j \left(c + \frac{1-c}{B(\alpha_j,\beta_j)} p_j^{\alpha_j - 1}(1-p_j)^{\beta_j - 1}\right)$

The likelihood for worker j:

$$\begin{aligned} L_j(p;t) &= \prod_i g_{ij}(t_i, p_j) \left(c + \frac{1-c}{B(\alpha_j,\beta_j)} p_j^{\alpha_j - 1}(1-p_j)^{\beta_j - 1}\right) \\ &= \frac{1-c}{B(\alpha_j,\beta_j)} p_j^{\alpha_j - 1 + \sum_i \mathbf{I}\{A_{ij}=t_i\}}(1-p_j)^{\beta_j - 1 + \sum_i \mathbf{I}\{A_{ij} \neq t_i\}} + c p_j^{\sum_i \mathbf{I}\{A_{ij}=t_i\}}(1-p_j)^{\sum_i \mathbf{I}\{A_{ij} \neq t_i\}} \end{aligned}$$

E-step:

Suppose $t_i \sim q_i$. Then:

$$\begin{aligned} Q_j(p) = \mathbb{E}_t\left[\ln L_j(p;t)\right] = \sum_t q(t) \Bigg[& \ln \frac{1-c}{B(\alpha_j,\beta_j)} + \left(\alpha_j - 1 + \sum_i \mathbf{I}\{A_{ij} = t_i\}\right) \ln p_j \\ & + \left(\beta_j - 1 + \sum_i \mathbf{I}\{A_{ij} \neq t_i\}\right) \ln(1-p_j) \\ & + \ln c + \left(\sum_i \mathbf{I}\{A_{ij} = t_i\}\right) \ln p_j + \left(\sum_i \mathbf{I}\{A_{ij} \neq t_i\}\right) \ln(1-p_j) \Bigg], \end{aligned}$$

which can be simplified to

$$Q_j(p) = \ln \frac{1-c}{B(\alpha_j,\beta_j)} + \ln c + (\alpha_j - 1)\ln p_j + (\beta_j - 1)\ln(1-p_j) + \left[\ln p_j \sum_i q(A_{ij}) + \ln(1-p_j) \sum_i q(-A_{ij})\right],$$

where

$$q_i(t_i) = \frac{\prod_j g_{ij}(t_i, p_j)}{\sum_{t_i} \prod_j g_{ij}(t_i, p_j)} \qquad (1)$$

M-step: Taking derivative with respect to p_j:

$$\frac{\partial Q_j}{\partial p_j} = \frac{\alpha_j - 1 + \sum_j q(A_{ij})}{p_j} + \frac{\beta_j - 1 + \sum_j q(-A_{ij})}{1 - p_j} = 0$$

which results in

$$p_j = \frac{\alpha_j - 1 + \sum_j q(A_{ij})}{\alpha_j + \beta_j - 2 + \sum_j q(A_{ij}) + \sum_j q(-A_{ij})}$$