Dynamic Filter: An Adaptive Algorithm for for Processing Data with the Crowd

Katherine Reed Austin Shin Beth Trushkowsky
Harvey Mudd College
kireed@hmc.edu, ashin@hmc.edu, beth@cs.hmc.edu

Abstract

A common operation done on databases is filtering: taking a set of items and finding the subset that satisfies certain restrictions. Crowdsourcing can be applied to evaluate subjective or complex restrictions. We divide the filtering process by separating a user's query into multiple restrictions that crowd workers can evaluate on each item. We then dynamically order those items and restrictions to reduce how many must be asked. We evaluate our technique using simulations based on data from Amazon's Mechanical Turk and present preliminary results.

Introduction

We are studying the efficient use of crowdsourcing to filter databases. The filtering process consists of applying a set of restrictions, called predicates, onto each item in a database that can be evaluated to either true or false. Items that satisfy all predicates are said to pass the filter.

We employ crowd workers to evaluate predicates that require them to reason subjectively and search for information. We use Amazon's Mechanical Turk (MTurk), a platform where requesters recruit workers to work on Human Intelligence Tasks (HITs). In each HIT, workers vote on whether an item is true or false for a given predicate.

The more difficult or subjective the predicate, the more votes we need to gather from workers in order to be certain that the majority answer is the correct answer. Thus harder or more ambiguous predicates lead to increased monetary cost and completion time. We hypothesize that the number of votes needed to process all items can be reduced by prioritizing predicates for which items are likely to be false, or in other words, prioritizing highly selective predicates. We define the selectivity of a predicate as the fraction of items that are false for that predicate. Prioritizing selective predicates reduces the number of votes because if an item is false for one predicate, it has already failed the filter and does not need to be evaluated with other predicates. However, we initially have no information about predicate selectivity, so our algorithm must learn and adapt as we gather votes from crowd workers.

In this work-in-progress, we describe our algorithm and evaluate how well it reorders predicates to reduce the cost of processing all items in the database. We test our algorithm with data from MTurk and discuss future work.

Overview of Data Flow

Database filtering can be thought of as items from a database flowing through a data processing pipeline where they are evaluated with various predicates to determine if they meet the restrictions of the filter. In our approach, we adapt the concept of an eddy (Avnur and Hellerstein 2000), a mechanism for dynamically rearranging parts of the pipeline. Items to be filtered are held in a queue outside the eddy. Each item flows into the eddy and is routed to a predicate, chosen by the algorithm described in the next section. A HIT is generated to ask an MTurk worker whether the item satisfies the predicate. The worker submits one vote for the item-predicate pair using a seven-point scale that captures a true or false value and a confidence level; the ends of the scale are "Yes (totally sure)" and "No (totally sure)". If there are not at least five votes for the item-predicate pair, the item returns to the queue.

Once five votes have been collected, we need to aggregate all votes for that item-predicate pair by estimating the certainty that the majority answer is the correct answer. We apply previous work on selective repeated-labeling (Sheng, Provost, and Ipeirotis 2008) to determine whether more votes are needed to reach consensus. If the uncertainty is high, the item is returned to the queue to collect two more votes. An item is done being processed once a predicate evaluates to false, or after all predicates are evaluated.

Adaptive Algorithm for Predicate Selection

To choose a predicate for an item, a lottery is held among the item's remaining predicates, with each predicate awarded a number of tickets proportional to its selectivity.

We do not know a priori which predicates are the most selective, so we estimate selectivity as the ratio of "false" votes to total votes for a given predicate. Our algorithm must continually update its estimates as more votes are entered. This process of choosing a predicate resembles a traditional Multi-Armed Bandit problem (Auer et al. 1995). We want to balance "exploiting" the predicate with the highest selectivity with "exploring" all the predicates. We found that the simple lottery system described above "explored" too much. To compensate, we use a multiplicative factor of two to increase the number of tickets for the most selective predicate, so that it is favored more strongly. This factor is a heuristic that we are continuing to explore to find an optimal strategy. We also "fast-track" an item to a predicate, bypassing the lottery, if its consensus uncertainty is below a threshold and the item is likely to be false.

Experimental Results

In this section, we describe how our adaptive algorithm performs with predicates of varying selectivity and show that it is more efficient than our baseline, a random ordering. The metric of interest is the number of HITs needed to process all the items.

For these initial results, we simulate the eddy process using votes gathered from MTurk workers. We first generated a database of 20 restaurants and devised a set of 10 predicates to filter restaurants with. We then collected a data set of votes by having workers evaluate the restaurant-predicate pairs, collecting 30 votes each. We researched the restaurants to determine ground truth.

We run our algorithm by simulating worker votes by sampling votes with replacement from the data gathered from MTurk. We set the uncertainty thresholds for vote consensus and fast-tracking to 0.15 and 0.3, respectively. When calculating uncertainty, we weight votes by workers' indicated confidence level.

Predicate	Selectivity
Does this restaurant serve Chinese food?	95%
Is this restaurant open until midnight?	95%
Does this restaurant have its own website?	5%

Table 1: The three predicates used in the simulation, along with their selectivities

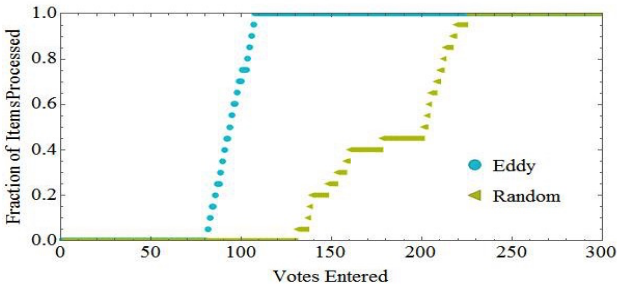

Figure 1: Fraction of items processed versus number of votes for the adaptive eddy algorithm and the random-ordering algorithm

Figure 1 shows the averaged results of 1000 simulation runs using the three predicates specified in Table 1 for our adaptive algorithm and a random-ordering algorithm. Our adaptive algorithm filtered all the items on average 130 votes quicker than the random algorithm. Of course, the number of votes may vary depending on the predicates used. Of the ten predicates, these three were the least ambiguous and therefore well suited to first investigate the impact of predicate selectivity. We compared predicate ambiguity using the average entropy: first computing the entropy across the 30 votes for each restaurant-predicate pair, and then averaging the entropy values grouped by predicate. Additional simulations with different predicate combinations, omitted due to space, also show the adaptive algorithm out-performing the random algorithm.

Next Steps

We have several plans for next steps. First, we want to gather more sets of votes from MTurk about restaurants for predicates of different selectivities and ambiguities to evaluate the generality of our initial results. We also plan to do a sensitivity analysis of the algorithm's parameters, as well as compare to an "optimal" algorithm that knows the true predicate selectivity (Parameswaran et al. 2012). Finally, we plan to run our algorithm live on MTurk, posting HITs and gathering votes from workers in real time.

References

Auer, P.; Cesa-Bianchi, N.; Freund, Y.; and Schapire, R.E. 1995. Gambling in a Rigged Casino: The Adversarial Multi-armed Bandit Problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society.

Avnur, R. and Hellerstein, J. M. 2000. Eddies: Continuously Adaptive Query Processing. In *Proceedings of SIGMOD*. ACM.

Parameswaran, Aditya, et al. 2012. Crowdscreen: Algorithms for filtering data with humans. In *Proceedings of SIGMOD*. ACM.

Sheng, V.; Provost, F.; Ipeirotis, P. 2008. Get Another Label? Improving Data Quality and Data Mining Using Multiple, Noisy Labelers. In *Proceedings of the 14th ACM SIGKDD*. ACM.