Eliminating Repeated Query Showing in Annotation System using Calculation of Reliability from Crowdsource

Peiyuan Zhao Kaishen Wang

1 Introduction

Cloud-based Open Lab for Data Science is a learning platform for big data education. A major challenge in data science education is how to allow students to experiment with algorithms for processing actual big data. Since the data is generally difficult to move around, making it unrealistic for students to download dataset and run experiments on local machine, it is necessary to build an experimental infrastructure on servers where the data sit. Such cloud-based lab would allow potentially a very large number of students to learn using real world big datasets, including those proprietary datasets that cannot be copied for sharing.

A prototype of the annotation part of this platform is already built by previous students in Prof. Zhai's group. The system allows researchers and students to upload their own IR algorithms and datasets. The engine then can produce ranked lists for annotators to perform manual annotation and relevance judgment.

Previously in the annotation system, every query will show up on every annotator's homepage, even if many annotators have already completed the annotation for it. Additional annotations on those queries are actually a waste of labor effort. However, what could be the threshold to indicate whether an assignment is completed or not, in case of annotators may annotate the same assignment with different results, is an interesting research topic. In this project, we make two big contributions:

- We rebuilt the whole annotation website, including both the frontend and backend. As for the frontend, we found and fixed many bugs and rewrote some logics which are using outdated libraries. As for the backend, we re-designed and re-implemented the schema and logic for performance reason. As the design and architecture of the annotation website is well explained in Guoqiao Li's Master Thesis[2], we will not illustrate too much about it.
- We developed and implemented an algorithm which could calculate the reliability of annotation from crowdsourcing in the annotation system based on the algorithm introduced in NIPS 2011[1].

2 Related Works

2.1 COLDS Overview

The basic idea of COLDS is to deploy software toolkits that contain algorithms for processing and analyzing big data sets on a

cloud-computing infrastructure. The learners can directly work with algorithms and their parameters to learn about the behavior of algorithms on real data sets that are available on the cloud.[2]
COLDS is beneficial in many different aspects. From education perspective, besides enabling big data programming assignments to be done at large scale, COLDS ensures the students learn skills that would be directly useful for solving an industry problem, thus minimizing the distance between education and applications in big data. From an industry partners perspective, contributing data sets to COLDS has multiple benefits, including visibility, training highly relevant labor force for the company, access to potential candidates for hiring, and annotating their data sets via crowd-sourcing annotations using annotation assignments. This last benefit would enable new research to be done in data science also, but requires a general annotation system to be included in COLDS.[2]

2.2 Annotation Subsystem

Annotation subsystem consists of two modules, which are instructor module and annotator module. Instructor module is responsible for 1). Uploading data set. 2). Creating and distributing new assignment. 3) Checking the submission from annotators. Annotator module is responsible for 1). Annotating documents based on given query, algorithm and parameters. 2) Free to use different query, algorithm and parameters to see how algorithms perform under different environments. Therefore, the instructor module is based three modules 1) upload module 2) search module 3) assignment module. The input of upload module is the name of the data set, the author of data set, the privacy of data set, and a list of .txt files. Search module is based on MeTA, the key functionality of search module is to send algorithm name and parameters to MeTA, retrieve result and output result. With assignment module, instructor create new assignments and distribute distribute assignments to annotators. The annotator module is based on these three modules too, but it also based on annotation module. Annotation module is responsible for collecting the judges made by annotator and make sure that instructor can see those judges.[2]

2.3 Problem Statement

In the original annotation system, every query will show up on every annotator's page. We want to stop showing a query to new annotators once we believe it has been well annotated. However, what could be the threshold for a query to be well annotated? This problem becomes more complex considering different annotators can make different choices for the query and they can have various accuracy history on past annotations. That is a typical crowdsourcing question and is the main problem we want to solve.

2.4 Naive Solutions and Limitations

Majority vote is a possible solution to finalize the correct answer for crowdsourcing systems. We can also use such method to find the correct answer and set up a threshold on the ratio of annotators submitting the correct answer. However, some lazy annotators may not carefully do the annotation tasks, but answer the questions with random choices. If most of their random choices happen to be identical or similar, we are getting the wrong answer.

3 Methods

3.1 Modeling

The paper [1] describes a very similar problem. We will use its model as our basis and make some modifications.

Every query includes m questions(documents) $\{t_i\}_{i\in[m]}$ as each being associated with an unobserved "correct" answer $a_i \in \{+1,-1\}$, where $+1$ indicates the document and the query is relevant and -1 indicates the irrelevancy. When a question(document) is assigned to an annotator, we get a possibly inaccurate answer from the annotator. We use $A_{ij} \in \{+1,-1\}$ to denote the answer if question(document) t_i is assigned to annotator s_j. As some annotators may be more careful than others and can annotate the assignments with better accuracies, we give a score $p_j \in [0,1]$ to each annotator to indicate their annotation reliability. If the question t_i is completed by annotator s_j, then

$$A_{ij} = \begin{cases} a_i & with\ \ probability\ \ p_j, \\ -a_i & with\ \ probability\ \ 1-p_j \end{cases}$$

and $A_{ij} = 0$ if t_i is not completed by s_j. We make the assumption that every query is independent and every document associated with the query is independent. We also assume that the annotators' performance will be consistent over all the questions. Although we will update their reliability scores after the completion of each query.

Finally, we model the problem with a bipartite graph G $(\{t_i\}_{i\in[m]} \cup \{s_j\}_{j\in[n]}, E)$, where each edge corresponds to a question-annotator assignment.

3.2 Existing Algorithms

An existing algorithm[1] describes a way to calculate the reliability of annotators and distinguish annotators with bad working experiences with others. We will develop our algorithm based on that and adapt it to the scenario of queries in annotation system.

The existing algorithm is an iterative algorithm, that can make inference of correct answer from crowdsourcing and update annotators' reliability scores. It operates on real-valued task messages $\{x_{i\to j}\}_{(i,j)\in E}$ and annotator messages $\{y_{j\to i}\}_{(i,j)\in E}$. The annotator messages are initialized as independent Gaussian random variables. At each iteration, the messages are updated according to the described update rule, where ϑ_i is the neighborhood of t_i. Intuitively, an annotator message $\{y_{j\to i}\}$ represents the belief on how 'reliableâĂŹ the annotator j is, such that the final estimate is a weighted sum of the answers weighted by each annotatorâĂŹs reliability: $\hat{a} = sign(\sum_{P_j\in\vartheta_i} A_{ij}y_{j\to i})$.

`Inference`
`Input:` $E, \{A_{ij}\}_{(i,j)} \in E, k_{max}$
`Output: Estimation` $\hat{a}(\{A_{ij}\})$
`1: For all` $(i,j) \in E$ `do`
`Initialize all` $y^{(0)}_{j\to i}$ `with`
↪ `Random` $Z_{ij} \sim N(1,1)$;
`2: For` k `= 1,..., ` k_{max} `do`
`For all` $(i,j) \in E$ `do`
↪ $x^{(k)}_{i\to j} \longleftarrow \sum_{j'\in\vartheta_i\setminus j} A_{ij'}y^{(k-1)}_{j'\to i}$;
`For all` $(i,j) \in E$ `do`
↪ $y^{(k)}_{j\to i} \longleftarrow \sum_{i'\in\vartheta_j\setminus i} A_{i'j}x^{(k)}_{i'\to j}$;
`3ïŽ For all` $i \in [m]$ `do`
↪ $x_i \longleftarrow \sum_{j\in\vartheta_i} A_{ij}y^{(k_{max}-1)}_{j\to i}$;
`4: Output estimate vector`
↪ $\hat{a}(\{A_{ij}\}) = [sign(x_i)]$;

3.3 Our Approach

We had to overcome several technical challenges in order to adapt the existing algorithm to our scenario.

- The existing algorithm doesn't output a single reliability score for each annotator (but have a vector of reliability scores over all the questions instead). We normalize the reliability scores for each annotator and output them.

- The existing algorithm doesn't use annotators' reliability score as the first priors in the calculation. To solve this, we develop the following strategy. If the annotators have never completed any assignments before, we will initialize it with 0.75. Otherwise, we will use their latest reliability scores as priors for calculation. The reason of choosing the value 0.75 is because we believe the worst case for an annotator is choosing the answer randomly, which should have 50% probability to be correct. Thus, we choose 0.75 as it is the mean of the lower and upper bound, which is 0.5 and 1.0 respectively.

- The existing algorithm doesn't worry about the number of annotators assigned to the tasks while that is our main goal. However, as we figure out how to update annotators reliability scores, we can make decisions on those scores. We believe a query is completed once the following requirements have been met: 1. There are at least TH1 annotators annotated it. This prevents the case of coincidence where a few annotators with high reliability scores don't do the assignment carefully but their not fully correct answers happen to be very similar. As TH1 gets higher, the possibility of such case happen decreases exponentially. 2. For every question in the assignment, we calculate the sum of the reliability score for the choice believed to be correct and record it as $S_{correct}$. For every question, the ratio of $S_{correct}$ to the total reliability scores for that question needs to be greater than a threshold TH2. This requirement guarantees that for every question, most people or majority of the authorities (high reliability score annotators) will hold the same opinion with the inference algorithm.

Here comes the full algorithm. For new annotators, we initialize their reliability scores with 0.75. This part is easy and intuitive, so we have no pseudo code for it. We want to have a modified version of the existing algorithm Inference. It is called ModifiedInference. In addition to the algorithm Inference, ModifiedInference will take annotators' latest reliability scores($\{P_j\}_{j\in[n]}$) and output the new values ($\{P'_j\}_{j\in[n]}$).

`ModifiedInference`
`Input:` $E, \{A_{ij}\}_{(i,j)\in E}, \{P_{j\,j\in[n]}\}, k_{max}$
`Output: Estimation` $\hat{a}(\{A_{ij}\}), \{P'_j\}_{j\in[n]}$

`1: For all` $(i,j)\in E$ `do` $y^{(0)}_{j\to i} \longleftarrow P_j$;
`2: For` k `= 1,..., ` k_{max} `do`
  `For all` $(i,j)\in E$ `do`
    $\hookrightarrow x^{(k)}_{i\to j} \longleftarrow \sum_{j'\in\vartheta_i\setminus j} A_{ij'} y^{(k-1)}_{j'\to i}$;
  `For all` $(i,j)\in E$ `do`
    $\hookrightarrow y^{(k)}_{j\to i} \longleftarrow \sum_{i'\in\vartheta_j\setminus i} A_{i'j} x^{(k)}_{i'\to j}$;
`3: For all` $i\in[m]$ `do`
  $\hookrightarrow x_i \longleftarrow \sum_{j\in\vartheta_i} A_{ij} y^{(k_{max}-1)}_{j\to i}$;
`4: For all` $j\in[n]$ `do`
  $\hookrightarrow P'_j \longleftarrow \frac{1}{m} * \sum_{i\in\vartheta_j} y^{(k_{max}-1)}_{j\to i}$;
`5: Output estimate vector`
  $\hookrightarrow \hat{a}(\{A_{ij}\}) = [sign(x_i)]$ `and`

↪ $\{P'_j\}_{j\in[n]}$;

We can decide if a query is well annotated or not and its final answers based on ModifiedInference function. A_{ij} is a matrix where its entry (i,j) is the answer that annotator j give for question i. Note that we only make changes to annotators' reliability scores {P} based on new values {P'} if the Decision function returns completed as true.

NSA
Input: TH1, TH2, $k_{max}, \{A_{ij}\}, \{P_j\}_{j\in[n]}$
Output: (boolean) completed,
↪ Estimation
$\hat{a}(\{A_{ij}\}), \{P'_j\}_{j\in[n]}$
1: if A.column_counts < TH1: //not
↪ enough annotators annotated
return completed = false; //
↪ other outputs are
↪ irrelevant
2: Build every edge $x_{i\to j}$ and $y_{j\to i}$
↪ for edge set E.($i \in [m]$ and
↪ $j \in [n]$)
3: $\hat{a}(\{A_{ij}\}), \{P'_j\}_{j\in[n]}$ =
↪ ModifiedInference(
↪ $E, \{A_{ij}\}, \{P_j\}_{j\in[n]}, k_{max}$)
4. $S_{total} \longleftarrow \sum_{j\in[n]} P_j$;
5: For all $i \in [m]$ do:
$S_{correct} \longleftarrow \sum_{j\in[n]\&\&A_{ij}==\hat{a}(A_{ij})} P_j$;
if $S_{total} * TH2 > S_{correct}$: return
↪ completed = false;
6ïŽ return completed = true and
↪ output $\hat{a}(\{A_{ij}\}), \{P'_j\}_{j\in[n]}$

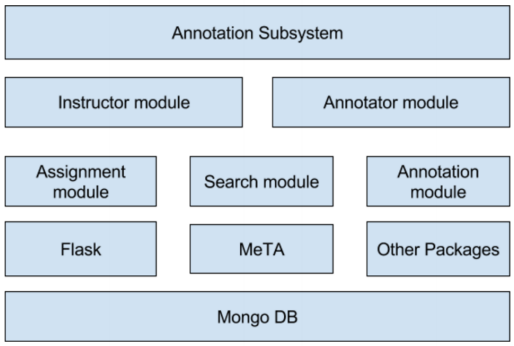
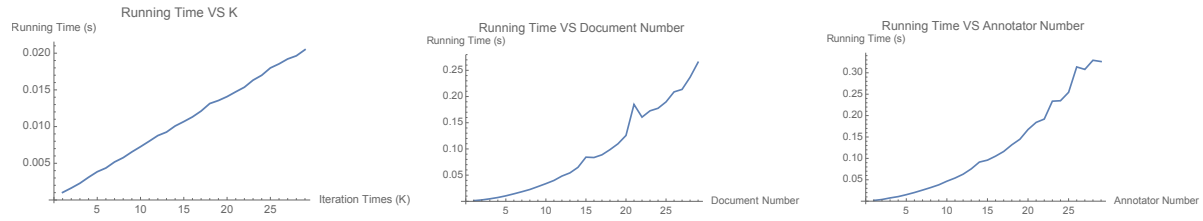


Figure 1: Annotation Subsystem Architecture

4 Implementation

4.1 Rewrite Annotation System

The detailed implementation can be found in Guoqiao's Thesis[2]. Although we rewrote the whole system, the architecture and framework we used are very similar. Due to the limited of space, many details are ommitted. Annotation subsystem is the fundamental supportive system of COLDS. As illustrated in Fig1, Annotation subsystem includes 2 major modules 1).Instructor module 2). Annotator module. These two modules are high level modules. They are based on other modules like search module, annotation module and assignment module. All those three modules are based on flask, which is a very powerful and lightweight python back end framework.[2] The front end interface is built with Bootstrap and traditional web programming language like html and css. Assignment Module, Search Module and Annotation Module are based on Flask, which is a lightweight back end framework. Flask provides user with a lot of powerful packages so that developer can build a server with simple and structured code.[2]



(a) 4 Documents and 4 Annotators (b) 4 Annotators and 10 Iterations (c) 5 Documents and 10 Iterations

Figure 2: Running Time Effected by Different Factors

4.2 Implementation of NSA Algorithm

NSA algorithm is the key idea to help the system determine whether a query has been well annotated based on the existing information. We implement the NSA algorithm in the utilities part in the backend. Every assignment will keep a list of the incomplete queries. The list will be initialized with all the instructor input queries. After an annotation has been submitted, a series of actions will be triggered. For each query, the annotation part will query the database with all the previous annotation results of that query from the same assignment along with the credibility scores of their annotators. NSA function will be called then to make the decision. If the return value shows the query has been well annotated, it will be removed from the showing list of that assignment. Meanwhile, for those annotators, their credibility scores will be updated using the values return by NSA function.

We decide not to simply replace their credibility scores with the new values for the following reason. If the NSA algorithm believes the query has been completed, it is highly possible that the iteration converges. As a result, the credibility scores will be changed rapidly. For example, if the answer of an annotator believes to be incorrect, his/her credibility score tends to be a very low value like 0. Certainly, we don't want to have that big change using only one query.

5 Results

A big concern about the NSA algorithm is its efficiency. As it will be deployed on the cloud, its performance becomes more important considering large number of concurrent users. We test its performance using a single of "2.7 GHz Intel Core i5" CPU (which should be less powerful than any cloud server). We confirmed that the running time increases linearly to the number of iterations, quadratically to the number of documents and annotators. We believe that in normal cases, the number of documents should not exceed 10 (otherwise the instructor can divide it into different assignments), the number of annotators and iteration number should also not exceed 10 (if the algorithm cannot converge by then, the query has probably not been completed yet). We find that with 10 documents, 10 annotators and 10 iterations, NSA function can be done in 0.13 seconds, showing that in most circumstances, the time cost of NSA should not be a problem.

6 Discussion

There is some future work that could be interesting. Firstly, the paper[1] provides and proves the performance of the algorithm it provides. It is possible to do the same to

our new algorithm. Secondly, we can find the empirically best value for all parameters (TH1, TH2 and k_{max}). Lastly, for a single query, the time cost of making the decision increases quadratically to the number of annotators. If many annotators have made different decisions on the query, the system should stop making the calculation but alert the instructor about it instead.

References

[1] Karger, D. R., Oh, S., and Shah, D. Iterative learning for reliable crowdsourcing systems. In *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 1953–1961.

[2] Li, Q. Design and implementation of the instructor module and annotator module in colds.