Quantitative Judgment Aggregation for Evaluating Contestants

Hanrui Zhang
Computer Science Department
Duke University
Durham, NC 27705
hrzhang@cs.duke.edu

Yu Cheng
Computer Science Department
Duke University
Durham, NC 27705
yucheng@cs.duke.edu

Vincent Conitzer
Computer Science Department
Duke University
Durham, NC 27705
conitzer@cs.duke.edu

Abstract

Quantitative judgment aggregation is a new research topic in (computational) social choice. In this model, agents judge the relative quality of different entities, and from this we aim to obtain an aggregate assessment of their relative qualities. As is the case for the more traditional voting-by-ranking model in social choice, there are corresponding statistical problems. For example, a race provides a "judgment" of the relative abilities of the contestants, and we may wish to aggregate the results of multiple races. In this context, we no longer need to worry about, e.g., strategic misreporting by the judges, which enlarges the space of possibilities for aggregation rules. We investigate this larger space theoretically and evaluate on data from various real races.

1 Introduction

In the theory of *voting*, each voter *ranks* a set of alternatives, and a *voting rule* maps the vector of rankings to either a winning alternative or an aggregate ranking of all the alternatives. There has been significant interaction between computer scientists interested in voting theory and the *learning-to-rank* community. This latter community is interested in problems such as learning how to rank webpages in response to a search query, or how to rank recommendations to a user (see, e.g., (Liu 2009)). Here, too, one may wish to aggregate multiple rankings into a single one, for example aggregating the ranking results from different algorithms ("voters") into a single meta-ranking. The interests of the communities differ somewhat: e.g., the learning-to-rank community is less concerned about strategic voting, and more about learning how much weight to assign to each voter. Still, a natural point of intersection for these two communities is a model in which there is a latent "true" ranking of the alternatives, of which all the votes are just noisy observations. Given such a model, it is natural to try to estimate the correct ranking given the votes—and a procedure for doing so corresponds to a voting rule. (See, e.g., (Young 1995; Conitzer and Sandholm 2005; Meila et al. 2007) for early work, and (Elkind and Slinko 2015) for a more recent overview.)

Voting rules are but one type of mechanism in the broader field of *social choice*, which concerns itself with the broader problem of making decisions based on the preferences and opinions of multiple agents, which are not necessarily represented as rankings. For example, in *judgment aggregation* (for an overview, see (Endriss 2015)), judges assess whether certain propositions are true or false. If we simply take a majority vote of the judges for each proposition, the resulting values for the propositions may be logically inconsistent with each other. This raises the question of what are good judgment aggregation rules that do result in logically consistent outcomes. The observation that there are other types of input that are aggregated in social choice leads to the natural question of whether these also have analogous problems in statistics and machine learning (as is the case for ranking).

In this paper, we study this question for a relatively new model in social choice, which is a *quantitative* judgment aggregation problem (Conitzer, Brill, and Freeman 2015; Conitzer et al. 2016). Specifically, the goal is to aggregate *relative quantitative judgments*, such as: "I believe that using 1 unit of gasoline is as bad as creating 2.7 units of landfill trash." As in the case of ranking, there are example where similar relative "judgments" are produced by a process other than an agent reporting them. Consider, for example, a race in which contestant A finishes in 20:00 and contestant B in 30:00. Hence, the "judgment" is that A is 1.5 times as fast as B. In a different race, their relative performance may be different. We may wish to aggregate these performances, in order to evaluate the contestants as well as to predict their relative performance in a future race. Given the different motivation, some of the aspects that are important in the social choice setting are less important to us here. For example, in a social choice setting, we may worry about agents strategically misreporting their judgments. In the contexts we are considering, this is not relevant because a race is not a strategic agent. As we will see, this will expand the set of rules that we may wish to consider.

Specifically, our goal is to obtain a *theoretically well motivated*, *transparent*, *interpretable*, and *auditable* aggregate measure of how relatively well contestants have performed. These other attributes, in addition to predicting future contests well, are essential for the responsible use of these techniques for, e.g., establishing a leaderboard.

1.1 Motivating Examples

In this section, we provide three motivating examples on which naïve mean/median performs poorly.

Example 1. When each race has some common "difficulty" (say, headwind, or how hilly a marathon route is), if a contestant only participates in the "easy" races (or only the "hard" races), simply taking the median or mean will return biased estimates, as shown in Figure 1.

Contestant \ Race	Boston	New York	Chicago
Alice	2:00:00	2:10:00	1:50:00
Bob	2:11:00	2:18:00	2:01:00
Charlie			2:09:00

Figure 1: Bob finished 8 minutes faster than Charlie in the same race (Chicago), which suggests that Bob runs marathons faster than Charlie. However, if we look at the results of all three races and simply take mean or median, Charlie's mean/median finishing time will be faster than Bob's. This is because, in this example, (it appears that) conditions were more favorable in the Chicago marathon.

Example 2. If our data shows that Alice has beaten before Bob in some race, and Bob has beaten Charlie in another race, but we have never seen Alice and Charlie competing in the same race, we may want to predict that Alice is a faster runner than Charlie (see Figure 2). More generally, if Bob does not show up in the testing dataset, then the naïve median/mean effectively ignores all the data on Bob. However, some of the information can be useful for comparing other contestants.

Contestant \ Race	Boston	New York	Chicago
Alice		2:10:00	
Bob	2:11:00	2:18:00	2:01:00
Charlie			2:09:00

Figure 2: The same results as in Figure 1, but with different data missing. If we only look at the data on Alice and Charlie, it is difficult to judge who is the faster runner; if anything, Charlie appears slightly faster. However, if we know Bob's results in these races, then we can use transitivity to infer that Alice is faster than Charlie.

Example 3. When the variance of the difficulty of the race is much higher than the variance in the contestants' performances, taking the median will essentially focus on the result of a single race (the race with median difficulty) as illustrated in Figure 3.

Quantitative Judgment Aggregation (QJA) takes care of all the issues above because it considers *relative* performance instead of absolute performance. The same is true for voting methods, but those only consider the ranking of the contestants in each race and discard the numerical information.

Contestant \ Race	Boston	New York	Chicago
Alice	2:00:00	2:10:00	1:50:00
Bob	2:11:00	2:18:00	2:01:00
Charlie	2:10:00	2:32:00	2:09:00

Figure 3: Simply taking the median throws away the useful information in the other two races if the common noise has high variance. Specifically, everyone's median race time is in Boston, so based on this we would predict Charlie to be faster than Bob. However, if we consider the other two races as well it certainly seems that Bob is faster than Charlie.

2 Quantitative Judgment Aggregation with ℓ_p-Loss

We are now ready to formally define Quantitative Judgment Aggregation (QJA). We generalize earlier definitions (Conitzer, Brill, and Freeman 2015; Conitzer et al. 2016) slightly by adding the exponent p; $p = 1$ corresponds to how it is defined in that earlier work.

Definition 1 (Quantitative Judgment Aggregation (QJA) with ℓ_p-Loss)**.** A pairwise contest result is a tuple (a, b, y), indicating that contestant a outperformed contestant b by y units in a contest. Fix $p > 0$. Given a set of contestants V with $|V| = n$, a set of pairwise contest results $E = \{a_i, b_i, y_i\}_{i=1}^m$, and weights $\{w_i\}_{i=1}^m$, the QJA problem with ℓ_p-loss asks for a vector $x \in \mathbb{R}^n$ that minimizes

$$\sum_{i=1}^{m} w_i |x_{a_i} - x_{b_i} - y_i|^p$$

Intuitively, the vector x reflects the estimated skill levels of the contestants, and serves as a predictor for the outcome of a future contest. c is by how much a actually outperformed b in one contest in our data, and the quantity $x_a - x_b$ is by how much we expect contestant a to outperform contestant b in a future contest. Generally speaking we will place the same weight $w_i = 1$ on all races.

QJA with ℓ_2-loss is in important ways similar to taking the average for each contestant.

Lemma 1. *When all contestants participate in all contests, and all weights $w_i = 1$, QJA with ℓ_2-loss is equivalent to taking the mean of each contestant's results.*

Proof. Consider any pair of contestants a and b and all $k = m/\binom{n}{2}$ pairwise contest results between a and b: $\{(a, b, y_i)\}_{i=1}^k$. Let $t_{a,i}$ and $t_{b,i}$ be the results of a and b in the i-th race, we have $y_i = t_{a,i} - t_{b,i}$.

Observe that $\sum_{i=1}^k (x_a - x_b - y_i)^2$ is minimized when $x_a - x_b = \frac{1}{k}\sum_i y_i$, and choosing $x_a = \frac{1}{k}\sum_i t_{a,i}$ and $x_b = \frac{1}{k}\sum_i t_{b,i}$ satisfies this condition. Thus, by setting x to be the mean vector of all contest results, we minimize the ℓ_2-loss between all pairs of contestants simultaneously. □

In contrast, QJA with ℓ_1-loss is closely related to taking the median. When there are only two contestants a and b, QJA with ℓ_1-loss consists in setting $x_a - x_b$ to the median of

the pairwise differences in races. This is considered an essential property in true social choice applications (Conitzer et al. 2016)—for one, it removes incentives to strategically misreport—which is why earlier work did not consider other values of p; however, for our applications here, this property is not essential.

2.1 MLE Interpretation of ℓ_p-Loss Rules

We next show that every ℓ_p-loss QJA rule can be interpreted as the maximum-likelihood estimator for the "true" relative abilities of the contestants, under a model in which contest results are noisy observations of these latent abilities.

Recall that the i-th pairwise race result (a_i, b_i, y_i) represents that player a_i performed y_i units better than player b_i. Assume there exists a latent ground-truth ability vector x^*, and the value of y_i is drawn independently for each i, from the distribution with $\Pr[y_i] \propto \exp(|y_i - (x^*_{a_i} - x^*_{b_i})|^p)$. For example, when $p = 2$, we observe the difference between the abilities of any pair of contestants with i.i.d. Gaussian noise. We now show that this specific family of noise models produces QJA with ℓ_p-loss as the MLE rule.

Proposition 1. *The ℓ_p-loss QJA rule is the MLE for $\Pr[y_i] \propto \exp(|y_i - (x^*_{a_i} - x^*_{b_i})|^p)$.*

Proof. The MLE selects a vector x that maximizes the likelihood: $\prod_i \exp(|y_i - (x_{a_i} - x_{b_i})|^p)$. After taking logarithm the likelihood becomes $\sum_i |y_i - (x_{a_i} - x_{b_i})|^p$, which is exactly the objective function of ℓ_p-loss QJA. □

3 Computational Aspects of Quantitative Judgment Aggregation

In this section, we study the computational complexity of the QJA problem with ℓ_p-loss for different values of p. We give a clean dichotomy result: we show that the problem is NP-hard when $0 < p < 1$, and can be solved in polynomial time when $p \geq 1$. We also present faster algorithms for the special cases $p \in \{1, 2\}$.

We first prove that the QJA problem is a special case of the ℓ_p-regression problem.

Definition 2 (ℓ_p-regression)**.** Given a matrix $A \in \mathbb{R}^{m\times n}$ and a vector $z \in \mathbb{R}^n$, find a vector x that minimizes $\|Ax - z\|_p^p$.

Theorem 1. *Fix any $p > 0$. The QJA problem with ℓ_p-loss is a special case of ℓ_p-regression.*

Proof. Given a QJA instance (V, E) with $|V| = n$, $E = \{a_i, b_i, y_i\}_{i=1}^m$ and weights $\{w_i\}_{i=1}^m$, we construct an ℓ_p-regression instance (A, z) as follows. The matrix A has dimensions $m \times n$; for the i-th row of A, let $A_{i,a_i} = w_i$, $A_{i,b_i} = -w_i$, and $A_{i,j} = 0$ for all other $j \notin \{a_i, b_i\}$. Let z denote the n-dimensional vector with $z_i = w_i y_i$. We conclude the proof by noting that any optimal solution x to the ℓ_p-regression instance (A, z) is an optimal solution to the original QJA instance (V, E). This is because x minimizes $\|Ax - z\|_p^p = \sum_{i=1}^m ((Ax)_i - z_i)^p = \sum_{i=1}^m w_i |x_{a_i} - x_{b_i} - y_i|^p$. □

3.1 Polynomial-Time Algorithms When $p \geq 1$

We first show that ℓ_p QJA admits polynomial-time algorithms when $p \geq 1$. Because QJA with ℓ_p-loss is a special case of ℓ_p-regression, it is sufficient to show that ℓ_p-regression can be solved efficiently. ℓ_p-regression is a fundamental problem with many applications in statistical data analysis and machine learning. It has been studied extensively (see, e.g., (Nesterov and Nemirovskii 1994; Xue and Ye 2000; Drineas, Mahoney, and Muthukrishnan 2006; Dasgupta et al. 2009; Shalev-Shwartz and Tewari 2011; Clarkson and Woodruff 2013; Meng and Mahoney 2013; Cohen and Peng 2015; Yang et al. 2016)). In particular, we invoke the following recent result from (Bubeck et al. 2018).

Lemma 2 ((Bubeck et al. 2018))**.** *Fix $p \geq 1$. Given $A \in \mathbb{R}^{m\times n}$ and $z \in \mathbb{R}^n$, we can compute a vector x such that*

$$\|Ax - z\|_p \leq \min_x \|Ax - z\|_p + \varepsilon$$

in $\widetilde{O}_p\left(\left(\mathrm{nnz}(A)\left(1 + m^{|\frac{1}{2}-\frac{1}{p}|}\sqrt{\frac{n}{m}}\right) + m^{|\frac{1}{2}-\frac{1}{p}|}n^2 + n^\omega\right) \cdot \log\left(\frac{\|z\|_2^p}{\varepsilon}\right)\right)$ time, where $\mathrm{nnz}(A)$ is the number of non-zero entries in A, and ω is the matrix multiplication exponent. [1]

Since the dependence on ε is $\log(1/\varepsilon)$ in Lemma 2, so ℓ_p-regression can be solved up to machine precision in polynomial time. We immediately have the following corollary:

Corollary 1. *For any $p \geq 1$, QJA with ℓ_p-loss can be solved in polynomial time.*

Despite the ongoing effort of improving the running time for ℓ_p-regression for general values of p, in practice it is common to choose $p = 1$ or $p = 2$ due to their conceptual simplicity and faster running time. In the next section, we present faster algorithms for QJA when $p \in \{1, 2\}$.

3.2 Faster Algorithms When $p \in \{1, 2\}$

In this section, we show that ℓ_p QJA admits faster algorithms when $p \in \{1, 2\}$. Recall that given $A \in \mathbb{R}^{m\times n}$ and y, the ℓ_p QJA problem is to minimize $\|Ax - z\|_p^p$ over $x \in \mathbb{R}^n$. It is a special case of ℓ_p-regression where each row of A has exactly two non-zero entries ($A_{i,a_i} = w_i$ and $A_{i,b_i} = -w_i$).

We can formulate ℓ_1-regression as a linear program (LP). For ℓ_1 QJA, (Zhang, Cheng, and Conitzer 2018) gave a faster algorithm than using general LP solver.

Lemma 3 ((Zhang, Cheng, and Conitzer 2018))**.** *The ℓ_1 QJA problem can be reduced to Minimum Cost Flow. Using the flow algorithm in (Gabow and Tarjan 1989), ℓ_1 QJA can be solved in time $\widetilde{O}(m^{1.5}\log(nW))$, where $W = \max_i w_i$.*

The ℓ_2-regression problem is also commonly known as the (linear) Least-Square regression problem. We exploit the special structure of ℓ_2 QJA, and relate it to the problem of solving Laplacian linear systems (Spielman and Teng 2004; Koutis, Miller, and Peng 2011; Cohen et al. 2014). Formally, we prove the following theorem.

[1] Throughout the paper, we write $\widetilde{O}(f)$ for $O(f \operatorname{polylog} f)$, and we use $O_p(f)$ to hide factors related to p.

Theorem 2. *QJA with ℓ_2-loss can be computed by solving a Laplacian linear system. In particular, using the Laplacian solver in (Cohen et al. 2014), QJA with ℓ_2-loss can be solved in time $\widetilde{O}(m\sqrt{\log n})$.*

Proof. We first show that ℓ_2-regression corresponds to solving a linear system. Let $f(x)$ denote the objective value of ℓ_2-regression at x. The goal is to minimize

$$\tfrac{1}{2}f(x) = \tfrac{1}{2}\|Ax - z\|_2^2 = \tfrac{1}{2}x^\top A^\top Ax - z^\top Ax + \tfrac{1}{2}z^\top z.$$

The derivative is $f'(x) = A^\top Ax - A^\top z$. Therefore, $f(x)$ is minimized when $x = (A^\top A)^{-1}A^\top z$.

Let $L = A^\top A$. Because A has only two non-zero entries in each row and they sum up to 0, we can view A as an edge-vertex incidence matrix and thus L is a graph Laplacian. [2]

Finally, we invoke the main result of (Cohen et al. 2014): given an $n \times n$ Laplacian matrix L with m non-zero entries, one can compute an ε-approximate solution to a linear system in L in time $O(m\sqrt{\log n}\log(1/\varepsilon))$. □

3.3 NP-Hardness When $p < 1$

We complement our positive results by the following hardness result (see full version of the paper for its proof).

Theorem 3. *For any $0 < p < 1$, it is* NP*-hard to approximate the ℓ_p QJA problem within a factor of $1 + \frac{c}{n^2}$ for some constant c.*

4 Experiments

In this section, we conduct experiments in which we make ordinal and quantitative predictions based on our aggregate evaluation of contestants. All experiments are done on a laptop with 8GB of memory and a 2.6 GHz Intel Core i5 CPU.

To evaluate QJA, we consider settings where: (1) contests are reasonably frequent, (2) the contests provide not only a ranking of the participants but also a numerical score, (3) the outcomes vary from one contest to the next, and (4) not every contestant appears in every contest. Observing scores of several consecutive contests, we try to make ordinal/quantitative predictions of the chronologically next contest using QJA and alternative methodologies.

We use the Min-Cost Circulation subroutine provided in the **LEMON** library[3] to implement ℓ_1 QJA (as in (Zhang, Cheng, and Conitzer 2018)). We use Sparse Least Squares Regression provided by **SciPy** [4] to implement ℓ_2 QJA.

Datasets.

- *Programming contests.* We use data from `codeforces.com`, a website hosting frequent online programming contests. In each contest, a score is given to each participant according to their performance. We train on up to 50 consecutive contests to predict the next one. Contests used for training and testing are all Division 1 contests (meaning only more skilled users can participate) ranging from "VK Cup 2016 - Round 1" to "Codeforces Round #467". The corresponding time interval is about 2 years. There are usually 500 to 1000 contestants in each contest.
- *NYC and Boston marathons.* We use data from `marathonguide.com`, which publishes results of all major marathon events. In each marathon, the score of a runner is her finishing time in seconds. The numbers of finishers of both marathons are usually very large (more than 20000). We take only the first 1000 finishers in each marathon for training and evaluation. Contests used for training and testing include all NYC and Boston marathons from 2013 to 2017 in chronological order (i.e. Boston 2013 followed by NYC 2013 followed by Boston 2014, etc.). We train QJA on up to 5 (corresponding to a time interval of about 2.5 years) consecutive contests.
- *SAT solver competitions.* We use data from `satcompetition.org`, which hosts annual competitions of SAT solvers. We focus on the main-track results of 2017. Each SAT instance is treated as a contest, and each solver as a contestant. Since most instances are solved by few solvers, we restricted our attention to instances where more than 90% of the solvers terminate. The score of a solver, when it terminates, is the time used in seconds. If a solver does not terminate, its score is the time limit. There are 32 contestants (solvers) and 46 qualifying contests (instances). Note that for the SAT solver dataset, all contestants (solvers) appear in each contest (instance).

Evaluation. For Codeforces and marathon datasets, contests are naturally ordered by the times at which they happen. In such cases, we use several consecutive contests to predict the immediately succeeding one. For each point in a figure, we average the results over 5 runs on different test contests. Specifically, suppose there are $m + n$ contests. For every $k = 1, \ldots, m$, we report the average accuracy over $n = 5$ runs. For the i-th run, we use the $(m + i)$-th contest as test set, and use the k immediately preceding contests as training data.

In the SAT solver competition dataset, contests (i.e., SAT instances) exhibit no chronological order. We use all contests but one as the training set to predict the remaining one, and report the average.

Benchmarks. We evaluate ℓ_1 and ℓ_2 QJA against the following benchmarks. Means and medians are like QJA in that they use numerical information, but unlike QJA in that they use absolute instead of relative numbers; i.e., they do not correct for the difficulty of the individual contest. Kemeny-Young and Borda are like QJA in that they are social-choice-theoretic methods that use relative rather than absolute performance, but unlike QJA in that they do not use numerical information, only the ranking of contestants.

[2] A graph Laplacian matrix L satisfies $L_{i,j} < 0$ for any $i \neq j$, and $L_{i,i} = \sum_{j \neq i} |L_{i,j}|$.

[3] See `lemon.cs.elte.hu/trac/lemon`.

[4] See `docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.lsmr.html`.

- *Means.* For every contestant in the training set, we take the mean of her scores in all training contests in which she participates. This quantitative aggregation also induces an ordinal aggregation, simply by sorting the means. Note that if all contestants appear in every contest (as in the SAT solver dataset), then ℓ_2 QJA is equivalent to taking means.
- *Medians.* For every contestant in the training set, we take the median of her scores in all training contests in which she participates. Medians more closely resembles ℓ_1 QJA.[5]
- *The Kemeny-Young rule (Kemeny 1959; Young and Levenglick 1978; Young 1988).* This is a voting method that takes multiple (partial) rankings of the contestants as input. We obtain these rankings from the training contests. It outputs ordinal predictions only; specifically, it outputs a ranking that minimizes the number of *disagreements* on pairs of contestants with the input rankings. This rule has a natural interpretation as estimating the "correct" ranking (Young 1988; 1995), and is again closely related to ℓ_1 QJA and medians.[6] Finding the Kemeny-Young outcome is known to be NP-hard. There are exact methods based on integer programming that can solve instances with up to hundreds of candidates in practice, but for larger-scale data (like the datasets we consider), those methods do not scale. Instead, we use an open-source heuristic solver[7]. Based on our tests on smaller instances, the heuristic solver almost always produces outcomes of the same or similar quality to the certifiably optimal ones.
- *The Borda rule.* The Borda rule is a voting rule that only uses rankings as input and only produces a ranking as output. We use a normalized version of the Borda rule. The i-th ranked participant in contest j receives $1 - 2(i-1)/(n_j - 1)$ points, where n_j is the number of participants in the contest. The aggregated ranking result is obtained by sorting the contestants by their total points. Borda can be viewed as a variant of means [8], and is therefore also closely related to ℓ_2 QJA.
- *Ratings* (for the Codeforces dataset only). Codeforces maintains ratings for all users, using a variant of the Elo rating system (Elo 1978). The ratings are calculated based on all previous contests, and are intended to be predictive. We use the ratings right before each contest to predict its result. Note that although the ratings are quantitative, they are not on a scale related to the actual scores of the contestants, so we only use them to provide ordinal predictions.

[5]For example, if there are only two contestants who participate in every contest, and one obtains the exact same score in every contest, then ℓ_1 QJA comes down to the median score of the other.

[6]To see the similarity, one may again consider the example with two contestants, one of which gets the same score every time. The outcomes of the three rules are all determined by the median score of the other contestant.

[7]See `numerical.recipes/whp/ky/kemenyyoung.html`

[8]Borda assigns scores to contestants linear in their ranks, and when every contestant participates in all contests, the output ranking is induced by the mean of the scores assigned.

Contestants to make predictions for. Not every contestant appears in every contest, and some contestants in the test contest may not appear in *any* of the training contests. Making predictions for such contestants only muddles the evaluation of our methods. On the other hand, suppose we make predictions for all the candidates that have appeared in *some* training set. Then, as the size of the training set increases, we make predictions for more participants who do not participate regularly, potentially worsening the accuracy. We therefore try to make predictions for the largest possible set of contestants that is meaningful and stable when the size of the training set grows: the set of contestants that appear in the testing set and the chronologically last training contest. For the Codeforces dataset, we have no less than 130, normally about 170 core contestants. For the marathon dataset, we have no less than 40, normally about 60 core contestants. For the SAT solver competition dataset, the number is always 32 since every contestant participates in every contest.

Multiplicative vs. additive differences. Our aim is to consider and predict *relative* performance of contestants, but this can be interpreted in multiple ways. In the experiments, we consider *additive* differences between contestants ("A finished 30 seconds before B"), instead of using log to translate to multiplicative differences ("A's finish time was 0.98 of B's"). For all our benchmarks, considering additive differences does not make any difference in how they are computed; but, for QJA, it does matter. Experimental results show that the choice affects ℓ_1 QJA to only a barely noticeable extent. This is perhaps not surprising because ℓ_1 QJA is more similar to median-based approaches that are invariant to any monotone transformation. In contrast, experiments show that working with additive differences makes ℓ_2 QJA more accurate. Hence, we only report results in the additive model.

Ordinal predictions. We first focus on ordinal predictions, which allows us to compare to benchmarks that only output a ranking. We measure the accuracy by the percentage of wrong pairwise predictions on core contestants in the test contest's ranking.

Figure 4 shows the accuracies of QJA and the benchmarks on the Codeforces dataset. The Figure 4a shows the error rates of all methods. Note that Codeforces ratings are based on the full history, and we cannot restrict it to a limited training set. Hence, its performance is a constant in the figure. In Figure 4b, we pairwise compare ℓ_1 QJA to the benchmarks most related to it, Kemeny-Young and medians, as well as to Codeforces ratings. Numbers below 0 indicate that ℓ_1 QJA performs better. In Figure 4c, we compare ℓ_2 QJA to the benchmarks most related to it, Borda and means, as well as to Codeforces ratings. The figure shows that training on 50 contests, both versions of QJA, Kemeny-Young, and means are competitive with Codeforces ratings. In addition, ℓ_1 QJA performs similarly to Kemeny-Young, and ℓ_2 QJA performs similarly to means.

Figure 5 shows the accuracies of QJA and the benchmarks on the marathon dataset. Figure 5a shows the error rates of

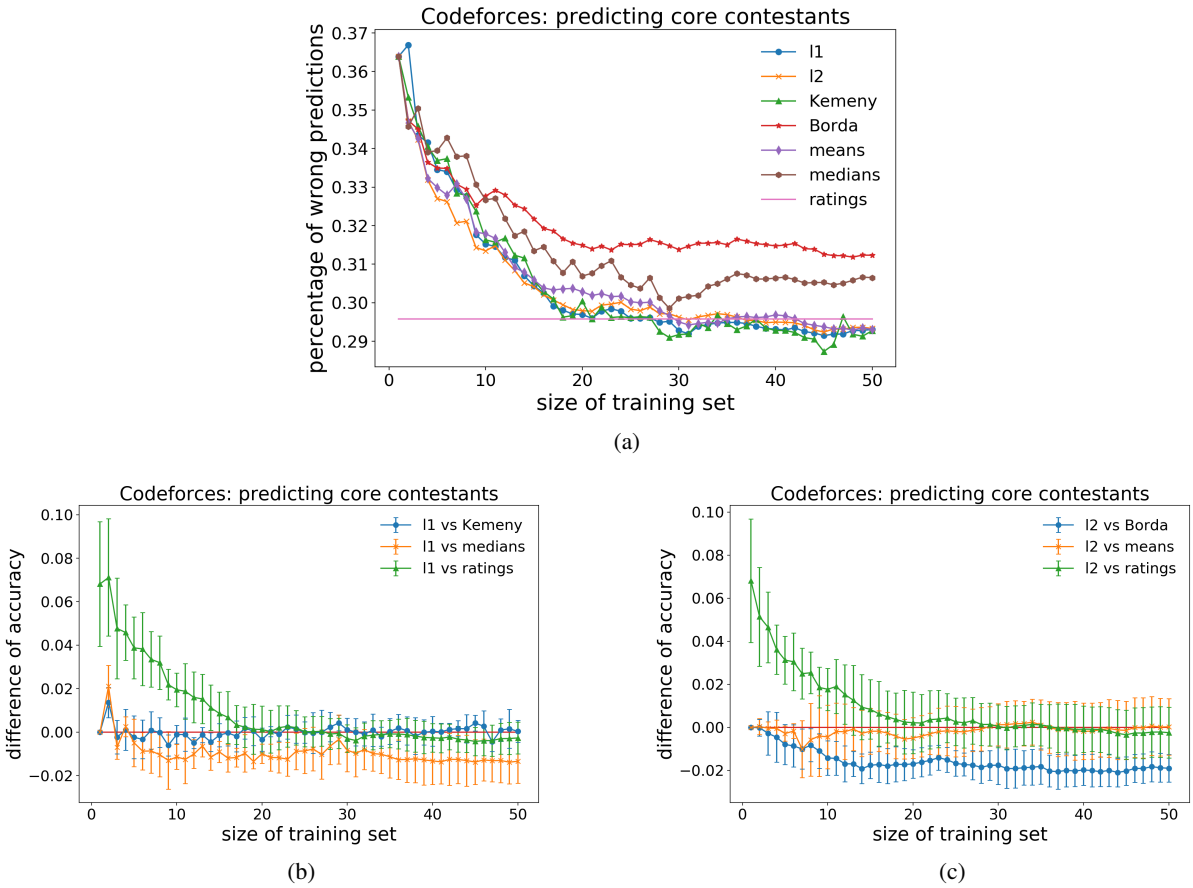


Figure 4: Comparison of accuracy on the Codeforces dataset.

all methods, and Figure 5b plots the differences between ℓ_1 QJA and Kemeny-Young, and between ℓ_2 QJA and means, respectively. Medians and Borda are not plotted in the right chart, since the gaps to QJA are already large enough in 5a.

The first two rows of Table 1 show the accuracies of QJA and the benchmarks on the SAT solver dataset, and the number of instances they "win"—that is, where they perform the best. The size of the training set here is always the same, consisting of all instances but one. There are 46 qualifying instances in total, as discussed above. ℓ_2 QJA does not appear separately since it is equivalent to means on this dataset.

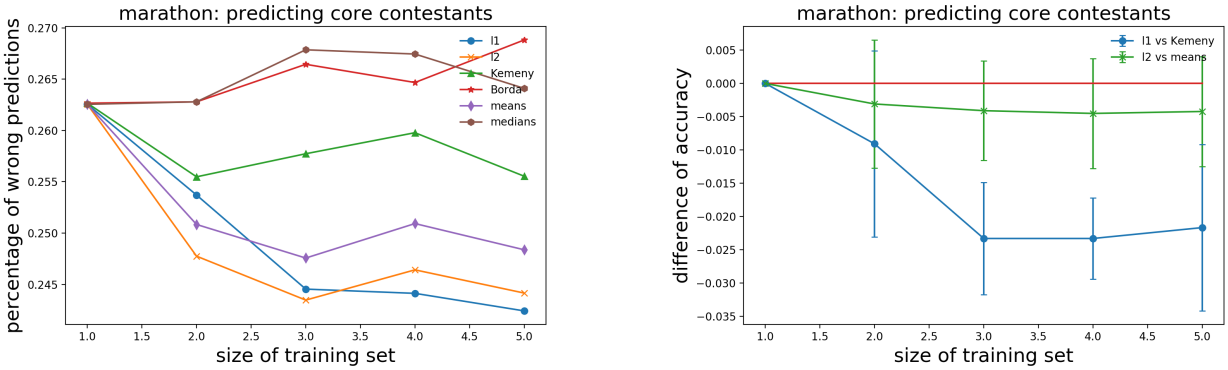
Quantitative predictions. We now turn our attention to quantitative predictions—where we aim to predict not only whether A finishes ahead of B, but also by how much. For this, we measure accuracy by the average absolute error between predicted pairwise differences of scores and the actual differences. We compare QJA only to means and medians, since the other benchmarks are designed only for ordinal predictions.

Figure 6 shows the errors of QJA and the quantitative benchmarks on the Codeforces dataset. Figure 6a shows the error rates of all methods. In 6b, we plot the differences of errors between (1) ℓ_1 QJA and medians, and (2) ℓ_2 QJA and means. QJA is significantly better than the benchmarks in terms of quantitative predictions. Figure 7 similarly compares QJA and the benchmarks on the marathon dataset.

The last two rows of Table 1 show the absolute errors of QJA and the benchmarks on the SAT solver dataset. ℓ_1 QJA beats the benchmarks on average, and wins more than half of the instances.

Remarks. From the experiments we see that the benchmark methods often perform very differently across datasets and tasks. For example, Kemeny-Young performs quite well on the Codeforces and SAT solver datasets, but much worse on the marathon dataset. Means gives good pairwise predictions on the Codeforces dataset, but performs less well on the other two. Also, despite its remarkable performance in pairwise predictions as a simple method, means is consistently worse than both versions of QJA in terms of quantitative predictions.

On the other hand, the two versions of QJA are never significantly worse than their benchmark counterparts, and in particular ℓ_1 QJA performs as well as or better than the best benchmark on all datasets, for both ordinal and quantitative predictions.

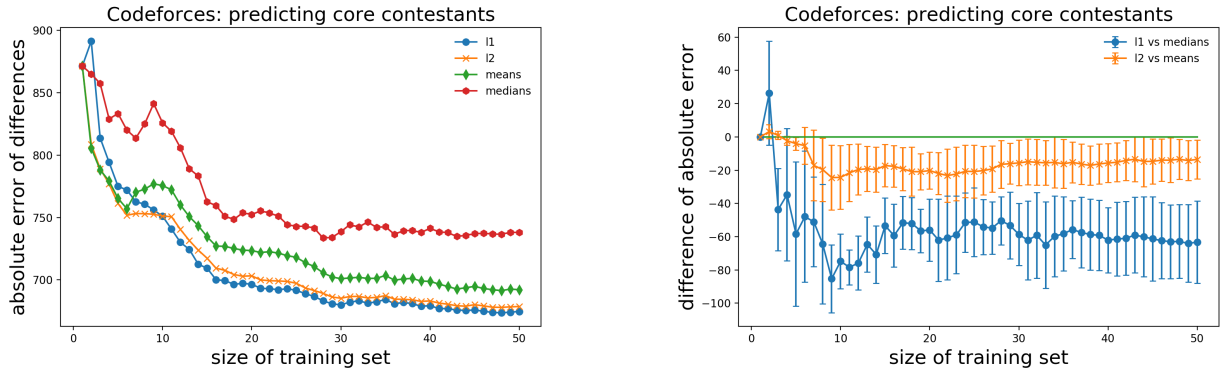


(a) (b)

Figure 5: Comparison of accuracy on the marathon dataset.

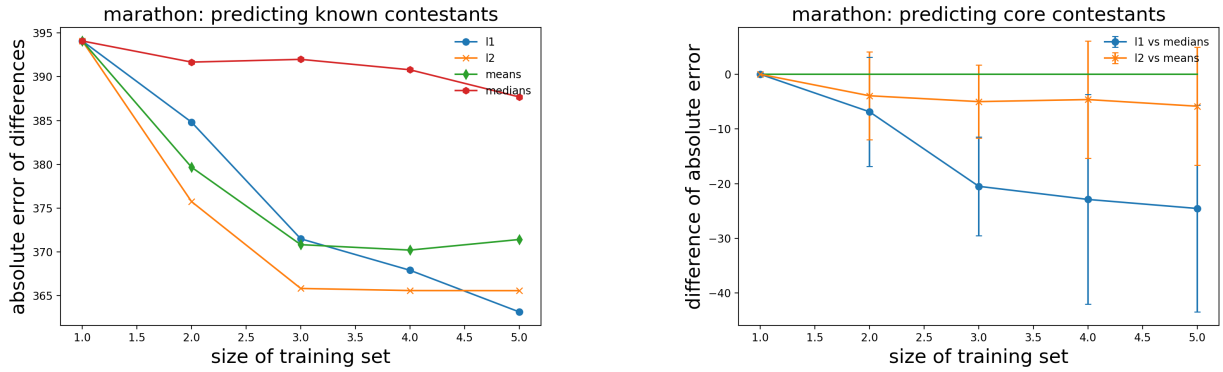
method	ℓ_1 QJA	Kemeny-Young	Borda	means	medians
average error rate	34.84%	35.98%	35.76%	38.25%	39.75%
# of contests won w.r.t. accuracy	11	11	0	14	10
average absolute error	327.17	N/A	N/A	372.85	340.95
# of contests won w.r.t. error	29	N/A	N/A	14	3

Table 1: SAT solver competition results.



(a) (b)

Figure 6: Comparison of absolute error of differences on the Codeforces dataset.



(a) (b)

Figure 7: Comparison of absolute error of differences on the marathon dataset.

References

Bubeck, S.; Cohen, M. B.; Lee, Y. T.; and Li, Y. 2018. An homotopy method for ℓ_p regression provably beyond self-concordance and in input-sparsity time. In *Proceedings of the 45th annual ACM Symposium on Theory of Computing (STOC)*, 1130–1137. ACM.

Clarkson, K. L., and Woodruff, D. P. 2013. Low rank approximation and regression in input sparsity time. In *Proceedings of the 45th annual ACM Symposium on Theory of Computing (STOC)*, 81–90. ACM.

Cohen, M. B., and Peng, R. 2015. ℓ_p row sampling by lewis weights. In *Proceedings of the 47th annual ACM Symposium on Theory of Computing (STOC)*, 183–192. ACM.

Cohen, M. B.; Kyng, R.; Miller, G. L.; Pachocki, J. W.; Peng, R.; Rao, A. B.; and Xu, S. C. 2014. Solving SDD linear systems in nearly $m \log^{1/2} n$ time. 343–352.

Conitzer, V., and Sandholm, T. 2005. Common voting rules as maximum likelihood estimators. In *Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 145–152.

Conitzer, V.; Freeman, R.; Brill, M.; and Li, Y. 2016. Rules for choosing societal tradeoffs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 460–467.

Conitzer, V.; Brill, M.; and Freeman, R. 2015. Crowdsourcing societal tradeoffs. In *Proceedings of the Fourteenth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 1213–1217.

Dasgupta, A.; Drineas, P.; Harb, B.; Kumar, R.; and Mahoney, M. W. 2009. Sampling algorithms and coresets for ℓ_p regression. *SIAM Journal on Computing* 38(5):2060–2078.

Drineas, P.; Mahoney, M. W.; and Muthukrishnan, S. 2006. Sampling algorithms for ℓ_2 regression and applications. In *Proceedings of the 17th annual ACM-SIAM Symposium on Discrete Algorithm (SODA)*, 1127–1136. SIAM.

Elkind, E., and Slinko, A. 2015. Rationalizations of voting rules. In Brandt, F.; Conitzer, V.; Endriss, U.; Lang, J.; and Procaccia, A. D., eds., *Handbook of Computational Social Choice*. Cambridge University Press. chapter 8.

Elo, A. E. 1978. *The rating of chessplayers, past and present*. Arco Pub.

Endriss, U. 2015. Judgment aggregation. In Brandt, F.; Conitzer, V.; Endriss, U.; Lang, J.; and Procaccia, A. D., eds., *Handbook of Computational Social Choice*. Cambridge University Press. chapter 17.

Gabow, H. N., and Tarjan, R. E. 1989. Faster scaling algorithms for network problems. *SIAM Journal on Computing* 18(5):1013–1036.

Kemeny, J. 1959. Mathematics without numbers. *Daedalus* 88:575–591.

Koutis, I.; Miller, G. L.; and Peng, R. 2011. A nearly-$m \log n$ time solver for SDD linear systems. 590–598.

Liu, T.-Y. 2009. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval* 3(3):225–231.

Meila, M.; Phadnis, K.; Patterson, A.; and Bilmes, J. 2007. Consensus ranking under the exponential model. In *Proceedings of the 23rd Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 285–294.

Meng, X., and Mahoney, M. W. 2013. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Proceedings of the 45th annual ACM Symposium on Theory of Computing (STOC)*, 91–100. ACM.

Nesterov, Y., and Nemirovskii, A. 1994. *Interior-point polynomial algorithms in convex programming*, volume 13. SIAM.

Shalev-Shwartz, S., and Tewari, A. 2011. Stochastic methods for ℓ_1-regularized loss minimization. *Journal of Machine Learning Research* 12(Jun):1865–1892.

Spielman, D. A., and Teng, S. 2004. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. 81–90.

Xue, G., and Ye, Y. 2000. An efficient algorithm for minimizing a sum of p-norms. *SIAM Journal on Optimization* 10(2):551–579.

Yang, J.; Chow, Y.-L.; Ré, C.; and Mahoney, M. W. 2016. Weighted SGD for ℓ_p regression with randomized preconditioning. In *Proceedings of the 27th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 558–569. SIAM.

Young, H. P., and Levenglick, A. 1978. A consistent extension of Condorcet's election principle. *SIAM Journal of Applied Mathematics* 35(2):285–300.

Young, H. P. 1988. Condorcet's theory of voting. *American Political Science Review* 82:1231–1244.

Young, H. P. 1995. Optimal voting rules. *Journal of Economic Perspectives* 9(1):51–64.

Zhang, H.; Cheng, Y.; and Conitzer, V. 2018. A better algorithm for societal tradeoffs. In *Seventh International Workshop on Computational Social Choice*.