KNEWTON DATA CHALLENGE

JOHN WANG

ABSTRACT. Premised on information theoretic results, I create an algorithm that attempts to maximize the entropy of a created ExamSet. Students are differentiated by weighted scores of the questions they answer correctly, where the weights are determined by the difficulty of the question. An ExamSet's entropy is defined by the probability of a student obtaining a given score.

First, this paper shows how I estimate the difficulty of a question (the probability r_j of answering question j correctly) using an iterative hill-climbing algorithm. Then, the paper shows how entropies of questions can be computed in order to approximate the entropy of an ExamSet. I show theoretical runtime results for a randomized algorithm to select questions based on entropy levels. Finally, I show how the results from this randomized algorithm are used as inputs into a genetic algorithm.

CONTENTS

1. Problem Formulation

Let there be n individuals taking an exam each year, each of whom take k of the K total questions available. We require that L of the K questions be offered to at least one student, and also that $0 < k < K$ be satisfied. In this paper, I will examine strategies for finding a ranking of students given a previous year's results as training data.

2. Measuring Question Difficulty

In order to do this, we want to formulate some measure of the difficulty of each question j. This motivates our examination of r_j, the probability that a student will get question j correct. To estimate r_j, one could naively use the sample mean from the training data for each question:

$$r_j \approx \frac{1}{n_j} \sum_{i=1}^{n} x_{ij} \tag{1}$$

Where x_{ij} denotes whether or not individual i answered question j correctly and n_j is the number of times question j was asked in the training data. This scheme works as long as the questions were assigned uniformly at random. However, as soon as there exists dependence among the questions, then this technique no longer works. If some set of questions q_1 are assigned to a group of students s_1 with higher probability than other questions and s_1 has a higher intelligence level than the average student, then r_j for $j \in q_1$ will be biased upwards.

Therefore, it is necessary to introduce a new variable θ_i which captures the intelligence level of student i. The probability that student i answers question j correctly will now have an additive factor of θ_i and will be given by $r_j + \theta_i$. If $r_j + \theta_i \geq 1$, then student i always answers j correctly, and if $r_j + \theta_i \leq 0$, then student i never answers question j correctly. One can see that $\theta_i = 0$ implies that student i is average, whereas $\theta_i > 0$ implies above average intelligence and vice versa for $\theta_i < 0$. Let $\gamma_{ij} = 1$ if student i was assigned question j and $\gamma_{ij} = 0$ otherwise. We will try to minimize the distance to the probability prediction:

$$\sum_{i=1}^{K} |x_{ij} - (r_j + \theta_i)|\gamma_{ij} \tag{2}$$

Using this distance function, we will attempt to estimate r_j for all j and θ_i for all i with the following algorithm. We start by initializing $\theta_i = 0$ for all i and $r_j = \frac{1}{n_j} \sum_{i=1}^{n} x_{ij}$ for all j. Now will will loop for T iterations. Suppose we are in iteration $t > 0$ and $t \leq T$. We will first update all intelligence values θ_i for all i by estimating $\sum_{j=1}^{K} |x_{ij} - (r_j + \theta_i)|\gamma_{ij}$. We will also estimate this with θ_i replaced with $\theta_i + \Delta_t$ and $\theta_i - \Delta_t$. The algorithm then replaces θ_i with $\theta_i - \Delta_t, \theta_i, \theta_i + \Delta_t$ depending on which one gives the smallest distance. We do this for all i and do the same for r_j by calculating $\sum_{i=1}^{n} |x_{ij} - (r_j + \theta_i)|\gamma_{ij}$ for $r_j - \Delta_t, r_j, r_j + \Delta_t$. We do this for all j and finish the iteration.

To calculate the distance offset Δ_t at each iteration t, we will use the function $\Delta_t = \frac{1}{2}e^{-t}$. This allows our distance to decrease at each iteration in an exponential manner. Intuitively this allows the algorithm to explore the sample space at the beginning and narrow down the search in later iterations.

To prove that the resulting r_j and θ_i will be good approximations of their actual values, we will show that our distance to the optimal result is bounded. Let r_j^* be the optimal value of r_j and θ_i^* be the optimal value of θ_i for all j and i. The distance using these values is given by:

$$E\left[\sum_{j=1}^{K} |x_{ij} - (r_j^* + \theta_i^*)|\gamma_{ij}\right] = round(k(r_j^* + \theta_i^*)) - k(r_j^* + \theta_i^*) \tag{3}$$

Here, $round(.)$ denotes the function that rounds . to the nearest integer. The expression follows because there will be k questions for which $\gamma_{ij} = 1$ for each student i, and since there can only be an integer number of $x_{ij} = 1$. Now, we will show that the above algorithm obtains values that are no worse than kn distance from the optimal. We compute:

$$\sum_{i=1}^{n}\sum_{j=1}^{K}|x_{ij}-(r_j+\theta_i)|\gamma_{ij} \quad \leq \quad \left|\sum_{i=1}^{n}\sum_{j=1}^{K}x_{ij}\gamma_{ij}\right| - \left|\sum_{i=1}^{n}\sum_{j=1}^{K}(r_j+\theta_i)\gamma_{ij}\right| \tag{4}$$

Using the triangle inequality. Since we always decrease our distance function as the iterations occur over time, all that needs to be shown is that the initial value is bounded by kn. We will use the initial values of r_j and θ_i, which means we can write the above expression as:

$$\begin{aligned}
\sum_{i=1}^{n}\sum_{j=1}^{K}x_{ij}\gamma_{ij} - \sum_{i=1}^{n}\sum_{j=1}^{K}\frac{1}{n_j}\sum_{i=1}^{n}x_{ij}\gamma_{ij} \quad &= \quad \sum_{i=1}^{n}\sum_{j=1}^{K}x_{ij}\gamma_{ij} - 2\sum_{n=1}^{n}\sum_{j=1}^{K}\frac{x_{ij}\gamma_{ij}}{n_{ij}} && (5)\\
&= \quad \sum_{i=1}^{n}\sum_{j=1}^{K}x_{ij}\gamma_{ij}\left(1-\frac{2}{n_{ij}}\right) && (6)\\
&\leq \quad \sum_{i=1}^{n}\sum_{j=1}^{K}x_{ij}\gamma_{ij} && (7)\\
&= \quad kn && (8)
\end{aligned}$$

From the fact that $n_{ij} > 0$ so that $(1-\frac{2}{n_{ij}}) < 1$. Therefore, we have bounded how far our algorithm can be from the optimal, since the optimal weights can only have a minimum distance of 0. In practice, however, $k(r_j^* + \theta_i^*)$ will not be an integer for all j and i which means that the distance for the optimal values of r_j^* and θ_i^* will be positive and proportional to n.

This analysis shows that we have found an approximation algorithm that allows us to find the difficulty of questions, r_j, while accounting for student's intelligence. We will now use r_{ij} as a probability measure for selecting the questions to use.

3. Ranking Students

Now, the process of ranking students will be discussed. Once a set of questions has been chosen, whether it is from the RPA algorithm or the genetic algorithm (to be discussed), the results of the test must be used to rank the students. This can be done in the following manner. Since we already know the probability of getting some question right, we compute a student's score σ_i using the following equation:

$$\sigma_i = \sum_{j=1}^{K}\frac{\gamma_{ij}x_{ij}}{r_j} \tag{9}$$

Basically, the score is a weighted sum of the number of questions that a student got right. Questions that have a lower probability of being answered correctly (low r_j) will have larger weights assigned to them. The total score of a student then is just the sum of the scores for each question the student was assigned, weighted by the difficulty of the question. Once σ_i has been computed for all i, the students can be ranked by σ_i.

4. Entropy and Question Selection

The main idea of the algorithms to be presented is to maximize the amount of entropy in the set of questions to be asked to students. We will call the pairing of all n students each with k questions to be an examset. The goal will then be to maximize the entropy of the examset. The reason to maximize entropy is because by doing this we maximize the amount of information available. Intuitively, we want to ask questions which will best separate good and bad students.

The information entropy will be a good measure of how separated the questions are. Thus, if we maximize the entropy of the examset, we will have the most information that is possible for ranking students. We will define the information entropy of an examset as:

$$\mathrm{H} = \sum_{l} P(l)\log\frac{1}{P(l)} \tag{10}$$

Where b is the size of the scoring bins and $P(l)$ is the probability that a given student's σ_i score will fall into bin l which covers scores in the range $[lb, l(b+1))$. Thus, we sum over all possible scores on a test and obtain the entropy for the entire examset. Maximizing entropy will intuitively produce exam scores which are as close to uniformly distributed as possible, which will provide the most information to use when ranking students. To compute $P(l)$, a dynamic programming solution will be used.

Notice that for each student, there are exponentially many possible scores, namely 2^k. For small k, such as $k = 5$, it is feasible to enumerate all scores. However, for large k, an approximation will be used. First we compute $p_i(s)$, the probability of getting s questions correct. One can use previously computed probabilities to compute new probabilities. Thus, given a student i and his k questions, we can compute $p_i(s)$ for $s \in [0, k]$ in the following manner. Let $p_i(s, q)$ be the probability of getting a score of s through the first q questions for student i, where $q \leq k$ and the questions are ordered arbitrarily. Then, we can compute $p_i(s, q)$ using the recursion:

$$p_i(s,q) = \begin{cases} p_i(s,q-1)(1-(r_q+\theta_i)) + p_i(s-1,q-1)(r_q+\theta_i) & \text{if } i \leq j \\ 0 & \text{else} \end{cases} \tag{11}$$

Assume that $p_i(s, q) = 0$ if $s \leq 0$ or $q \leq 0$ so that the $p_i(s-1, q-1)(r_q + \theta_i)$ term falls away if $s - 1 \leq 0$. The recursion follows because the student could either get question q right or wrong. The probability of $p_i(s, q)$ is then the probability that the student i had a score of s previously on question $q - 1$ times the probability of getting q wrong, i.e. $p_i(s, q-1)(1-(r_q+\theta_i))$, plus the probability of having a score of $s-1$ previously on question $q - 1$ times the probability of getting q right, i.e. $p_i(s - 1, q - 1)(r_q + \theta_i)$.

Initializing $p_i(s, 0) = 1$ and progressively computing $p_i(s, 1)$ for all $s \in [0, k]$, $p_i(s, 2)$, etc. will provide $p_i(s, k) = p_i(s)$.Thus, it is possible to find $p_i(s)$ for all $s \in [0, k]$ in $O(k^2)$ runtime. Once $p_i(s)$ are found, the approximation takes the average weight obtained from each question $\frac{1}{k}\sum_{j=1}^{k}\frac{1}{r_j}$ and uses that as the weight of having answered one extra question correctly. Using this subroutine to compute $p_i(s)$, it is possible to compute the evaluation function H in $O(nk^2)$ time.

4.1. **Greedy Approximation.** For the first attempt at a solution to this maximization problem, a greedy solution will be proposed. Since it is possible to calculate the entropies of each question, and it is likely that high entropy questions individually lead to high entropy totals, the greedy algorithm will calculate the top entropies for all questions and select the highest L. From there, a variety of methods will be developed to actually assign the questions to students.

Note that this is only an approximation to the optimal because the highest entropy questions individually do not necessarily provide the highest entropy examset since we are looking for entropy of scores. Thus, high entropy questions will contribute significantly to a higher entropy total score for each student, but it may be possible that questions are dependent, in which case a particular combination of questions could lead to very high entropy.

To compute the entropy H_j of each question j, we use the equation:

$$\mathrm{H}_j = r_j \log\frac{1}{r_j} + (1-r_j)\log\frac{1}{1-r_j} \tag{12}$$

Since r_j has already been calculated for each question j, calculating H_j is straightforward. Sorting the questions and taking the top L requires $O(K \log K)$ time. Now, a number of different strategies for assigning questions to students is employed.

4.2. **Randomized Probabilistic Assignment.** For the randomized probabilistic assignment (RPA) algorithm, probabilities will be given to each question and the questions for each student will be chosen randomly based on the probabilities provided. The crux of the matter is assigning probabilities which lead to valid examsets of high entropy. Assigning very large probabilities to the highest entropy questions means that there is a lower probability that the eventual examset is valid (i.e. has more than L questions assigned to at least one person). If an assignment is not valid, the algorithm will try again to reassign all questions. Keeping the expected number of retries constant will prevent the algorithm from having too large of a runtime.

To do this we will have a probability of assignment ρ_j to question j which is proportional to its entropy H_j. However, we will scale the largest and smallest probability ρ_j to differ by less than a multiplicative factor of c. Formally, we will set ρ_j for all j such that $c\min_j \rho_j \geq \max_j \rho_j$. Since these are probabilities, we also require that $\sum_j \rho_j = 1$ for all j in the subset of questions to be assigned to students. We shall add some constant x to each H_j so that $\frac{\mathrm{H}_{max}+x}{\mathrm{H}_{min}+x} \leq c$. We can solve for x and so that $x = \frac{H_{max}-H_{min}}{c-1}$. Let $\mathrm{H}'_j = \mathrm{H}_j + \frac{H_{max}-H_{min}}{c-1}$ so that the minimum is always within a multiplicative factor of c of the maximum. We will have:

$$\rho_j = \frac{\mathrm{H}'_j}{\sum_j \mathrm{H}'_j} \tag{13}$$

We will show that with the correct c, it is possible to keep the expected number of retries to a constant T. To do this, we note that the probability that all the questions in the set are assigned is 1 minus the probability that some question is not assigned. Moreover, we know that the probability that at least one question is not assigned is less than the probability that all questions are unassigned. This means:

$$\begin{aligned} Pr[\text{some question unassigned}] &\leq \sum_{j=1}^{L}(1-\rho_j)^n && (14)\\ Pr[\text{all questions assigned}] &\geq 1-\sum_{j=1}^{L}(1-\rho_j)^n && (15) \end{aligned}$$

Therefore, the expected number of tries t to get an iteration where all questions were assigned satisfies:

$$E[t] \leq \frac{1}{1-\sum_{j=1}^{L}(1-\rho_j)^n} \tag{16}$$

Thus, as long as $\sum_{j=1}^{L}(1-\rho_j)^n$ is small enough, the algorithm won't have to try too many attempts. Now, if we have $c\min_j \rho_j \geq \max_j \rho_j$, we can bound the expected number of tries at a constant. The largest possible value of $\sum_{j=1}^{L}(1-\rho_j)^n$ given the above constraint comes when all but one of the questions have the minimum probability $\min_j \rho_j$ of being selected and the last one has probability $c\min_j \rho_j$. Let ρ_{max} be that maximum probability, then we know that $\rho_{max}\left(\frac{L-1}{c}+1\right) = 1$. This means that $\rho_{max} = \frac{c}{L+c-1}$ and $\rho_{min} = \frac{1}{L+c-1}$. We obtain the probability:

$$\begin{aligned} \left(1-\frac{c}{L+C-1}\right)^n + \sum_{j=1}^{L-1}\left(1-\frac{1}{L+c-1}\right)^n &\leq \sum_{j=1}^{L}\left(1-\frac{c}{L+c-1}\right)^n && (17)\\ &= \sum_{j=1}^{L}\left(\frac{L-1}{L+c-1}\right)^n && (18)\\ &= L\left(\frac{L-1}{L+c-1}\right)^n && (19)\\ &\leq \frac{L^{n+1}}{(L+c-1)^n} && (20) \end{aligned}$$

To make the expected number of tries less than some constant T, we want to satisfy $E[t] \leq T$, which means we can solve for c:

$$\begin{aligned} \frac{1}{1-\frac{L^{n+1}}{(L+c-1)^n}} &\leq T && (21)\\ \frac{(L+c-1)^n}{(L+c-1)^n-L^{n+1}} &\leq T && (22)\\ (L+c-1)^n &\leq T((L+c-1)^n-L^{n+1}) && (23) \end{aligned}$$

Simplifying further, we see that we can set c such that it satisfies the condition below in order to get an expected number of tries less than T:

$$c \quad \leq \quad \left(\frac{TL^{n+1}}{T-1}\right)^{1/n} - L \tag{24}$$

For example, if the provided training data is used to calibrate c, and we want an expected number of tries of $T = 3$, then we have $c \leq 200.0$. Therefore, as long as the minimum ρ_j is kept at least c away from the maximum ρ_j, the number of expected tries can be bounded. Moreover, the markov inequality shows that the probability that the the number of tries is greater than the G is given by:

$$Pr[t \geq G] \leq \frac{E[t]}{G} \tag{25}$$

Therefore, the probability that the number of tries exceeds the expected number drops off linearly as the excess increases.

4.3. **Genetic Algorthim.**

The other algorithm that will be implemented is a genetic search algorithm which uses sampling to quickly compute H. The λ examsets from iteration t in the algorithm will be used as the parents of new examsets to be generated in iteration $t+1$. At each iteration, Λ new examsets will be created by mutation and crossover.

First, there will be some mutation rate μ_m and a crossover rate μ_c. These will govern the probability of mutating a student's assignment of questions. There will therefore be an expected $\mu_m nk$ mutations and $\mu_c nk$ crossovers. Each question will have some probability of mutatation, and the algorithm will go through all questions in an examset, seeing if a question needs to be mutated or crossed.

If a question is to be mutated, a new question is selected from the set of all questions using the probability assigment algorithm from RPA (notice that the question set will be all possible questions and we will extend the RPA probability assignment algorithm to all questions). If a question is to have a crossover, we will randomly select another examset with probability proportional to the examset's entropy, and we will select an individual within that examset with probability proportional to the individual's entropy. The crossover will switch the individual's set of questions for the two examsets.

All of the mutations and crossovers will be performed during an iteration, and this will be performed until there are a total of Λ new examsets spawned from the original λ parents. The mutation rate μ_m and crossover rate μ_c will decrease as the genetic algorithm proceeds, much like in simulated annealing, so that the sample space is explored initially.

The genetic algorithm uses the output of the greedy assignment algorithm as the initial examset. From there is mutates the examset with a mutation rate of 0.25, and proceeds to iterate on each examset.