Optimal Number of Choices in Rating Contexts

Sam Ganzfried, Florida International University SCIS, sganzfri@cis.fiu.edu

Overview

In many settings people give numerical scores to entities from a small discrete set. For instance, attractiveness from 1-5 on dating sites and papers from 1-10 for conference reviewing. We study the problem of understanding when using a different number of options is optimal. We study several natural processes for score generation. One may expect that using more options always improves performance, but we show that this is not the case, and that using fewer choices -- even just two -- can surprisingly be optimal. Our results suggest that using fewer options than typical could be optimal in certain situations. This would have many potential applications, as settings requiring entities to be ranked by humans are ubiquitous.

Figure 1: Hot or Not users rate attractiveness 1–10.

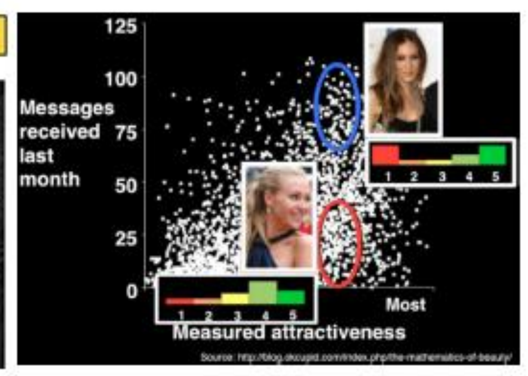


Figure 2: OkCupid users rate attractiveness 1–5.

Figure 3: Tinder users rate attractiveness 1–2.

Figure 4: IJCAI reviewers rate papers -5–5.

Model

Users have underlying integral ground truth score for each item in {1,…,n} and are required to submit an integral rating in {1,…,k}, for k << n.

Two generative models:

1. Uniform: the fraction of scores for each value from 1 to n is chosen uniformly at random (by choosing a random value for each and then normalizing)
2. Gaussian: the scores chosen according to a Gaussian distribution with a given mean and variance

We then compute ``compressed'' score distribution by mapping each full score s from {1,…,n} to {1,…,k} by applying s ← floor(s / (k/n)). We compute the average "compressed" score a_k and its error $e_k = |a_f - [(n-1)/k] * e_k|$, where a_f is the ground truth average score. The goal is to pick $\text{argmin}_k e_k$.

Theoretical characterization

Suppose scores are given by continuous pdf f (with cdf F) on $(0, 100)$, and we wish to compress them to two options, $\{0, 1\}$. Scores below 50 are mapped to 0, and scores above 50 are mapped to 1.

The average of the full distribution is

$$a_f = E[X] = \int_{x=0}^{100} x f(x)dx.$$

The average of the compressed version is

$$a_2 = \int_{x=0}^{50} 0 f(x)dx + \int_{x=50}^{100} 1 f(x)dx = \int_{x=50}^{100} f(x)dx = F(100) - F(50) = 1 - F(50).$$

So $e_2 = |a_f - 100(1 - F(50))| = |E[X] - 100 + 100F(50)|$.

For three options,

$$\begin{aligned} a_3 &= \int_{x=0}^{100/3} 0 f(x)dx + \int_{x=100/3}^{200/3} 1 f(x)dx + \int_{x=200/3}^{1} 2 f(x)dx \\ &= F(200/3) - F(100/3) + 2(1 - F(200/3)) = 2 - F(100/3) - F(200/3) \end{aligned}$$

$$e_3 = |a_f - 50(2 - F(100/3) - F(200/3))| = |E[X] - 100 + 50F(100/3) + 50F(200/3)|$$

In general for n total and k compressed options,

$$\begin{aligned} a_k &= \sum_{i=0}^{k-1} \int_{x=\frac{ni}{k}}^{\frac{n(i+1)}{k}} i f(x)dx \\ &= \sum_{i=0}^{k-1} \left[i \left(F\left(\frac{n(i+1)}{k}\right) - F\left(\frac{ni}{k}\right) \right) \right] \\ &= (k-1)F(n) - \sum_{i=1}^{k-1} F\left(\frac{ni}{k}\right) \\ &= (k-1) - \sum_{i=1}^{k-1} F\left(\frac{ni}{k}\right) \end{aligned}$$

$$\begin{aligned} e_k &= \left| a_f - \frac{n}{k-1} \left((k-1) - \sum_{i=1}^{k-1} F\left(\frac{ni}{k}\right) \right) \right| \\ &= \left| E[X] - n + \frac{n}{k-1} \sum_{i=1}^{k-1} F\left(\frac{ni}{k}\right) \right| \qquad (3) \end{aligned}$$

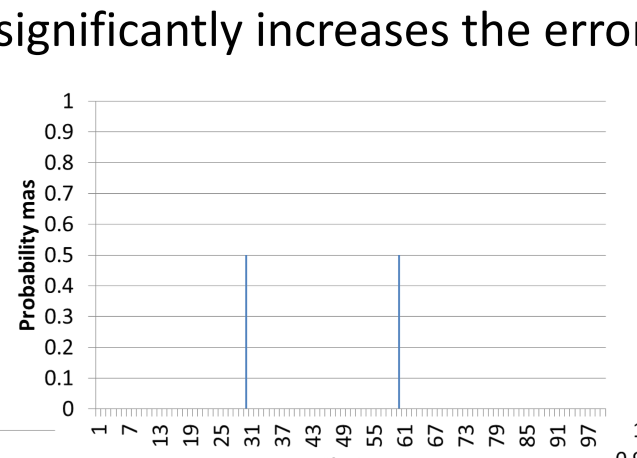
Equation 3 allows us to characterize the relative performance of choices of k for a given distribution f. For each k the characterization requires only knowing k statistics of f (the $k-1$ values of $F\left(\frac{ni}{k}\right)$ plus $E[X]$). In practice these could likely be closely approximated from historical data for small values of k.

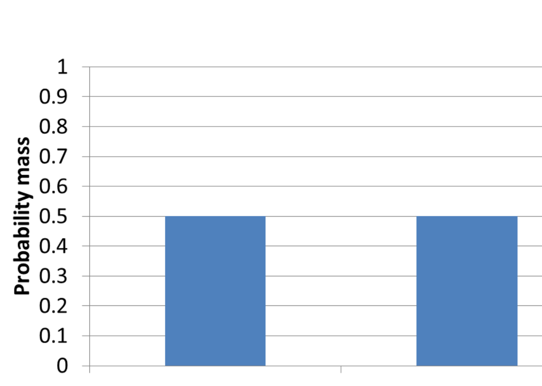
Example where k=2 outperforms k=3

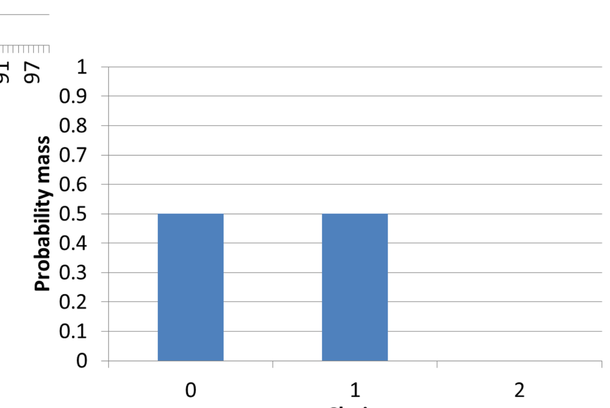
As an example we see that $e_2 < e_3$ iff

$|E[X] - 100 + 100 F(50)| < |E[X] - 100 + 50F(100/3) + 50F(200/3)|$.

$a_f = E[X] = 0.5 * 30 + 0.5 * 60 = 45$. If we use k = 2, then the mass at 30 will be mapped down to 0 (since 30 < 50) and the mass at 60 will be mapped up to 1 (since 60 > 50). So $a_2 = 0.5 * 0 + 0.5 * 1 = 0.5$. Using normalization of n/k = 100, $e_2 = |45 - 100 (0.5)| = |45 - 50| = 5$. If we use k = 3, then the mass at 30 will also be mapped down to 0 (since 0 < 100/3); but the mass at 60 will be mapped to 1 (not the maximum possible value of 2 in this case), since 100/3 < 60 < 200/3. So again $a_3 = 0.5 * 0 + 0.5 * 1 = 0.5$, but now using normalization of n/k = 50 we have $e_2 = |45 - 50 (0.5)| = |45 - 25| = 20$. So, surprisingly, in this example allowing more ranking choices actually significantly increases the error.







Computational simulations and analysis

For our simulations we used n = 100, and considered k = 2,3,4,5,10, which are popular and natural values. For the Gaussian model we used s = 1000, μ = 50, σ = 50/3. For each set of simulations we computed the errors for all considered values of k for m = 100,000 ``items'' (each corresponding to a different distribution generated according to the specified model). The main quantities we are interested in computing are the number of times that each value of k produces the lowest error over the m items, and the average value of the errors over all items for each k value.

Algorithm 1 Procedure for generating full scores in uniform model

```
Inputs: Number of scores n
  scoreSum ← 0
  for i = 0 : n do
    r ← random(0,1)
    scores[i] ← r
    scoreSum = scoreSum + r
  for i = 0 : n do
    scores[i] = scores[i] / scoreSum
```

Algorithm 3 Procedure for generating full scores in Gaussian model

```
Inputs: Number of scores n, number of samples s, mean μ, standard deviation σ
  for i = 0 : s do
    r ← randomGaussian(μ, σ)
    if r < 0 then
      r = 0
    else if r > n − 1 then
      r ← n − 1
    ++scores[round(r)]
  for i = 0 : n do
    scores[i] = scores[i] / s
```

Algorithm 2 Procedure for compressing scores

```
Inputs: scores[], number of total scores n, desired number of compressed scores k
  Z(n, k) ← n/k                                        ▷ Normalization
  for i = 0 : n do
    scoresCompressed[⌊i / Z(n,k)⌋] += scores[i]
```

	2	3	4	5	10
Uniform number of victories	5564	9265	14870	16974	53327
Uniform average error	1.32	0.86	0.53	0.41	0.19
Gaussian number of victories	3025	7336	14435	17800	57404
Gaussian average error	1.14	0.59	0.30	0.22	0.10

Table 1: Number of times each value of k in {2,3,4,5,10} produces minimal error and average error values, over 100,000 items generated according to both generative models.

	2	3
Uniform number of victories	36805	63195
Uniform average error	1.31	0.86
Gaussian number of victories	30454	69546
Gaussian average error	1.13	0.58

Table 2: Number of times each value of k in {2,3} produces minimal error and average error values, over 100,000 items generated according to both generative models.

	2	10
Uniform number of victories	8253	91747
Uniform average error	1.32	0.19
Gaussian number of victories	4369	95631
Gaussian average error	1.13	0.10

Table 3: Number of times each value of k in {2,10} produces minimal error and average error values, over 100,000 items generated according to both generative models.

	2	10
Uniform number of victories	32250	67750
Uniform average error	1.31	0.74
Gaussian number of victories	10859	89141
Gaussian average error	1.13	0.20

Table 4: Number of times each value of k in {2,10} produces minimal error and average error values, over 100,000 items generated according to both models. For $k = 10$, we only permitted scores between 3 and 6 (inclusive). If a score was below 3 we set it to be 3, and above 6 to 6.

	2	10
Uniform number of victories	93226	6774
Uniform average error	1.31	0.74
Gaussian number of victories	54459	45541
Gaussian average error	1.13	1.09

Table 5: Number of times each value of k in {2,10} produces minimal error and average error values, over 100,000 items generated according to both generative models. For $k = 10$, we only permitted scores between 3 and 7 (inclusive). If a score was below 3 we set it to be 3, and above 7 to 7.

Example where k=2 significantly outperforms k=10

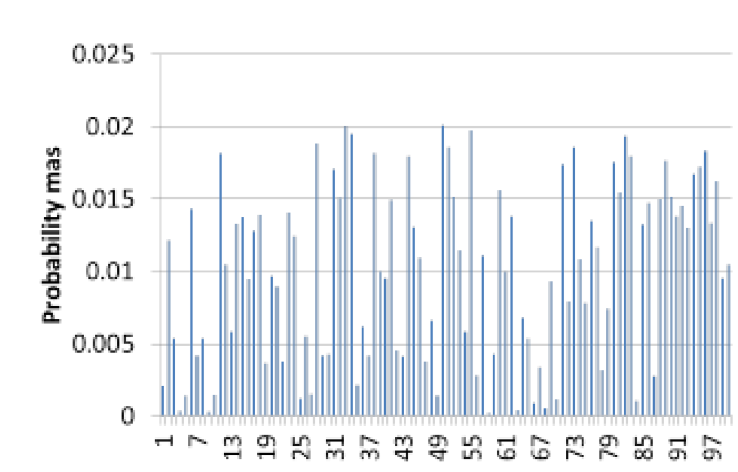


Figure 8: Example distribution for which compressing with $k = 2$ produces significantly lower error than $k = 10$. The full distribution has mean 54.188, while the $k = 2$ compression has mean 0.548 (54.253 after normalization) and the $k = 10$ compression has mean 5.009 (55.009 after normalization). The normalized errors between the means were 0.906 for $k = 10$ and 0.048 for $k = 2$, yielding a difference of 0.859 in favor of $k = 2$.

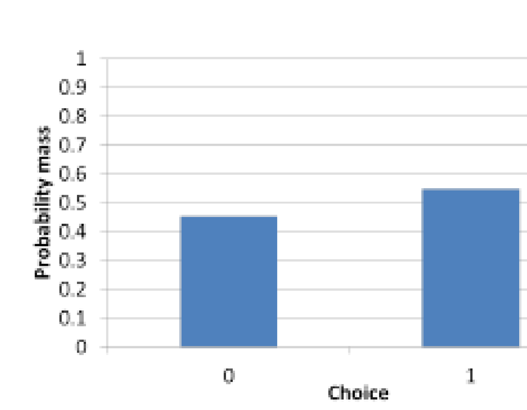


Figure 9: Compressed distribution for $k = 2$.

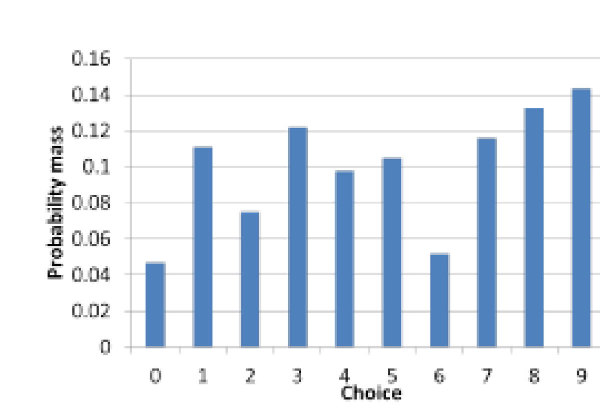


Figure 10: Compressed distribution for $k = 10$.

Related work

The most closely related work [Dubey10] studies the impact of using finely grained numerical grades (e.g., 100, 99, 98) vs. coarse letter grades (e.g., A, B, C). They conclude that if students care primarily about their rank in class (relative to the other students), they are often best motivated to work by assigning them to coarse categories (letter grades) than by the exact numerical exam scores. In a specific setting of ``disparate'' student abilities they show that the optimal absolute grading scheme is always coarse. Their model is game-theoretic; each player (student) selects an effort level, seeking to optimize a utility function that depends on both the relative score and effort level. Their setting is quite different from ours in many ways. For one, they assume that the underlying ``ground truth'' score is known, yet may be disguised for strategic reasons. In our setting the ultimate goal is to approximate the ground truth score as closely as possible.