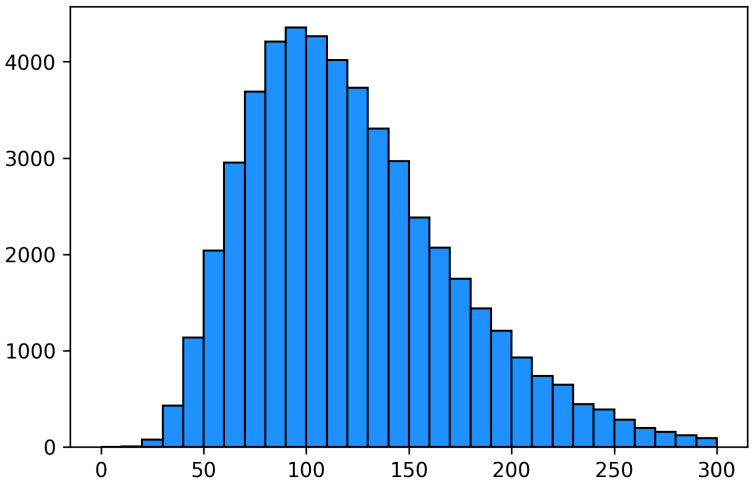
Analysis of the Prediction Tournament Paradox

by

V.M. van der Eng

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Wednesday June 25, 2025 at 15:00.



Student number:	5873800
Project duration:	April 2025 - June 2025
Thesis Comittee:	Dr. J. Söhl
	Dr. H. Schuttelaars
Supervisors:	Dr. J. Söhl

TU Delft

Abstract

In a prediction tournament, contestants are tasked with predicting the distribution of a random variable. To determine which contestant makes the most accurate predictions, scores are assigned based on the outcomes of the random variables. The scoring rules are designed such that a contestant's expected score decreases as their predicted values approach the true distribution. This implies that the contestant with the lowest score should be the most accurate predictor. However, simulation results show that this is not the case.

In this report, we found that for the common case of Bernoulli random variables, the true success probabilities affect the distribution of winners: it has a positive effect when the probability is closer to 0 or 1, and a negative effect when it is near 0.5. We also found that this distribution is not affected by whether contestant errors are drawn from a continuous distribution with fixed variance σ^2 or are simply $+\sigma$ or $-\sigma$.

Furthermore, contestants who make extreme predictions (always predicting 0 or 1) do not outperform those who predict values close to the true success probability. While the choice of scoring rule does influence the distribution of winners, it does not eliminate the paradox. We found that the the Pseudospherical and Power score with parameter β close to 1, and the Logarithmic score performed the best.

We extend our analysis to random variables with multiple categories. To support this extension, we introduce a new sampling method that builds on the one used in earlier simulations. In the binary model, we only needed one success probability for each random variable, but now we need multiple per random variable, while making sure that the sum of all the probabilities is exactly 1. Using a statistical distance, we determine how to model contestant predictions. For these random variables, we also analyze various scoring rules. In this case, we found that both the Pseudospherical score and the Power score, with β slightly larger than 1, and the Logarithmic score performed the best across various numbers of categories.

Similarly, we extend our analysis to continuous random variables. Because of time constraints, we only look at Normal distributions with known variance. We use the same statistical distance as for the multi-categorical random variables, the total variation distance, to determine how to model contestant predictions. We again look at several scoring rules and found that the Power and Pseudospherical scoring rules for values of β close to 1 and the Logarithmic score, performed the best in this scenario.

Layman's summary

Companies often host prediction tournaments in which contestants are asked to estimate how likley it is that a specific event occurs. For example, the chance of rain or no rain. Contestants are scored using a system designed to reward accurate forecasts. However, simulations reveal that these scoring systems do not succeed in identifying the most accurate forecasters. In this report, we investigate factors that may contribute to this discrepancy between scoring system that reward accuracy, but fail to identify the most accurate forecaster.

We found that when success probabilities are near 50%, this discrepancy worsens: worse contestants more frequently achieve the best score. We also found that making extreme predictions (always 0 or 1) does not give contestants an advantage, and changing how we model the deviation from the true probability for each contestant does not influence the discrepancy at all. We considered several different scoring systems and found that they can influence the severity of the discrepancy, but none eliminate it entirely.

We then extended our analysis beyond simple binary (yes/no) outcomes to more complex scenarios involving multiple categories or continuous outcomes, such as those drawn from uniform or normal distributions. For example, a multi-category prediction might involve estimating the likelihood of no rain, light rain, or heavy rain, while a continuous prediction could involve forecasting the likelihood of the number of milliliters of rainfall. We again tested several different scoring systems and found that they can influence the strength of the discrepancy to various degrees, but cannot eliminate it.

When combining the results for both the simple yes/no case and the multiple categories case, we found that the so called Power and Pseudospherical scores with parameter β slightly larger than 1, and the Logarithmic score reduced this discrepancy the most.

Contents

1 Introduction

In recent years, prediction tournaments have become more common as a means to find accurate forecasters. An example of such a forecasting tournament is the Good Judgment Project, a geopolitical prediction competition in which the top 2% of contestants are awarded the title of "superforecaster," along with other benefits (Tetlock & Hardner, 2015). Similar competitions exist in algorithmic settings as well. For instance, Netflix offered $1,000,000 to the team whose machine learning algorithm best predicted users' movie ratings. Kaggle has also hosted machine learning competitions involving the prediction of labels for data points (Witkowski, Freeman, Vaughan, Pennock, & Krause, 2022). What all these tournaments have in common is that only the winner (or the top few percent) are rewarded, but how fair is this?

In 2019, David Aldous described a paradox concerning these prediction tournaments in his article *A Prediction Tournament Paradox*, that questions the fairness of such tournaments (Aldous, 2019). In his tournament, contestants were tasked with predicting the success probability p of a Bernoulli random variable. To determine which contestant makes the most accurate predictions, scores are assigned based on the outcomes of the random variables. The Brier score was used for this purpose and this scoring rule is designed in such a way that a contestant's expected score decreases as their predicted values get closer to the true probability p (Brier, 1950). This suggests that the contestant with the lowest score should be the most accurate predictor. However, simulation results show that the most accurate predictor usually does not win the tournament. The exact details of this paradox are explained in Aldous' article and a brief overview is given in Section 2.

Understanding the prediction tournament paradox and the factors that influence it, is important for ensuring fairness in prediction competitions, like the ones mentioned previously. This report aims to analyze this paradox and the factors that contribute to it. Witowski et al have addressed this paradox as well, but analyzed several methods of selecting a winner in a way that incentivizes accurate forecasting (Witkowski et al., 2022), which is not what we will be doing in this report. Aldous discussed the effects of several aspects of the tournament model. He analyzed the effect of different error parameters for contestants. He looked a two methods of modeling the deviation from the true probability of contestants: $\pm$ a set deviation and a continuous uniform random variable. He also looked at the top 10 finishers and the effect of systematic over- and under-estimation (Aldous, 2019). In this report, we do a deeper analysis on aspects already discussed by Aldous. We also look at several other aspects, such as the effect of the distribution of the true success probabilities and the use of different proper scoring rules. There is a lot of literature on the properties of scoring rules, but not necessarily in the context of tournaments. Besides this, Aldous only models a tournament with Bernoulli random variables. We extend this model to multi-category random variables and continuous random variables and analyze several factors of this new model.

Because the scores of contestants are highly correlated and thus difficult to predict, we will make use of simulations to do our analysis. In general, we will simulate 4000 tournaments unless specifically mentioned otherwise. All simulations are done in Python and the code used can be found in Appendix C.

In Section 2, we begin by discussing the tournament model introduced by Aldous in his article (Aldous, 2019). Then, in Section 3, we dive into the literature on proper scoring rules. We will examine several popular scoring rules used for both discrete and continuous random variables, analyzing properties such as divergence and symmetry. None of this will be new work, only a summary of the existing literature.

In Section 4, we will begin our own analysis by taking a closer look at the tournament model introduced by Aldous. We analyze the scores and finishing positions of contestants, as well as the distribution of ranks among the top n finishers.

After this, we adjust different components of the original tournament model to analyze their effect on the paradox. In Section 5, we examine how altering the true success probabilities of the Bernoulli random variables influences the outcome. This is done by shifting fixed success probabilities toward the edges and the center of the interval $[0, 1]$, as well as by sampling success probabilities from various continuous distributions.

Besides the distribution of the true success probabilities, we can also vary the distribution of contestants' predictions. In the original simulations, each contestant is assigned an error (or standard deviation) and predicts the true success probability plus or minus that error. We expand on this by allowing the error for contestants' predictions to be drawn from a continuous distribution that depends on the standard deviation associated with a contestant. Aldous used a uniform distribution for this purpose in his article (Aldous, 2019). In Section 6, we extend this approach by analyzing predictions drawn from a beta distribution with various parameters. At the end of Section 6, we also consider contestants who make extreme predictions: always choosing 0 or 1 and thus predict the events directly instead of their probabilities.

The third element that can be modified in our prediction tournament is the scoring rule used. In the original simulation, the Brier score is applied, but many other proper scoring rules exist, as discussed in Section 3. We aim to analyze whether these scoring rules have a positive, negative, or no effect on the paradox. In this section, we will examine the logarithmic, spherical, pseudospherical, and power scores. The last two of these scoring rules include a parameter β, and we will also explore the effect of varying this β.

In all the previously mentioned sections, we focused exclusively on very simple random variables: those of Bernoulli type. In practice, however, we are likely to encounter other types of random variables, so we also consider their effect on the paradox. This is addressed in Sections 8 and 9, where we examine multi-categorical random variables and continuous random variables with known densities. To accommodate these types, some aspects of the original tournament must be adjusted. To maintain a comparable level of error for contestants as in the original model, we employ a statistical distance metric. Several options exist for this, and we will discuss and select the most suitable one. Changing the type of random variables also opens the possibility for using scoring rules that were not applicable in the binary case. Since the scoring rules used in the binary setting can often be adapted to other types of random variables, we will evaluate whether their performance changes under these new conditions.

Altogether, we seek to gain a deeper understanding of this prediction tournament paradox and what factors influence it. This knowledge is useful for contestants participating in such tournaments, who want to adopt a prediction strategy that makes them more likely to win (which may involve predicting less accurately), but also for the companies hosting such prediction contests. Knowing what factors cause worse predictors to win tournaments more often or what factors cause the opposite can be taken into account during the design of future forecasting competitions to ensure a fairer competition that incentives accurate forecasting.

2 A prediction tournament paradox

A prediction tournament consists of three parts:

1. events;
2. contestants;
3. a scoring rule;

For simplicity, we consider only binary events for now. Each event occurs with a probability p. Before the event takes place, contestants are asked to predict this probability p. After the events have occurred, we assign each contestant a score, using the following method:

Suppose that a contestant predicted the probability q for a single event. Then

$$\text{score} = (1-q)^2 \text{ if the event happened; score} = q^2 \text{ if not}$$

The total score of a contestant is the sum of the scores from all events. The objective is to achieve the lowest total score among all contestants.

We want to analyze whether the contestant with the lowest score is actually the best forecaster. To do this, we first look at the expected value of a contestant´s total score S. Let X denote the score of a contestant for a single event.

Then $\mathbb{P}(X = (1-q)^2) = p$ and $\mathbb{P}(X = q^2) = 1-p$. So

$$\begin{aligned} \mathbb{E}[X] &= p \cdot (1-q)^2 + (1-p) \cdot q^2 \\ &= p - 2pq + pq^2 + q^2 - pq^2 \\ &= p - p^2 + p^2 - 2pq + q^2 \\ &= p(1-p) + (q-p)^2 \end{aligned} \tag{2.1}$$

Let p_i be the true probability of event i and q_i be the probability predicted by the contestant. Then

$$\mathbb{E}[S] = \sum_i p_i(1-p_i) + \sum_i (q_i - p_i)^2 \tag{2.2}$$

We observe that the term $\sum_i p_i(1-p_i)$ is the same for every contestant and only depends on the true event probabilities. The term $\sum_i (q_i - p_i)^2$, however, depends on the contestant´s predictions q_i, and is the minimized when the predicted values q_i are close to the true probabilities p_i. This indicates that contestants are incentivized to make accurate predictions.

However, the actual score is not equal to the expectation, but contains some chance variation.

To examine the impact of this variability, David J. Aldous set up a simple simulation. We consider a tournament of 100 events. These events occur with probabilities $0.05, 0.15, 0.25, ..., 0.95$, with each probability appearing 10 times.

We look at 300 contestants. Each contestant has an error parameter σ, which is evenly distributed over an interval of length 0.3. If the true probability of an event occurring is p, then a contestant will predict either $p + \sigma$ or $p - \sigma$, each with equal probability and truncated to [0, 1]. Thus, a smaller σ corresponds to a more accurate forecaster.

Figure 2.1 shows the distribution of the ranks of the tournament winner for $0 < \sigma < 0.3$. We observe that the 100th most accurate contestant is most likely to win the tournament, while the most accurate contestant almost never wins. Figure 2.2 show the results for simulations with $0.05 < \sigma < 0.35$, $0.1 < \sigma < 0.4$ and $0.15 < \sigma < 0.45$. The same effects occur, just to a lesser degree.

An explanation for this phenomenon lies in the mean-variance trade-off. In the case of Figure 2.1, contestants with σ around 0 are always making roughly the same predictions. However, contestants that have a σ around 0.1 are making slightly more varying predictions. While their expected scores are worse, some will, by chance, have predictions that align well with outcomes, resulting in unusually low scores.

Aldous also showed that this result is not limited to just the winner of the tournament. Figure 2.3 shows a comparison between the ranks of the tournament winner and the top 10 finishers in a simulation with $0.15 < \sigma < 0.35$. We see that even though in both cases the winners/top 10 are most likely to come from the most accurate 10 forecasters, a large portions of the winners/top 10 do not come from the top 10 most accurate forecasters (Aldous, 2019).

As the effect of the paradox is the largest for $0 < \sigma < 0.3$, it is of interest to analyse what happens to the top 10 finishers for σ between 0 and 0.3. We do this in Section 4.3.

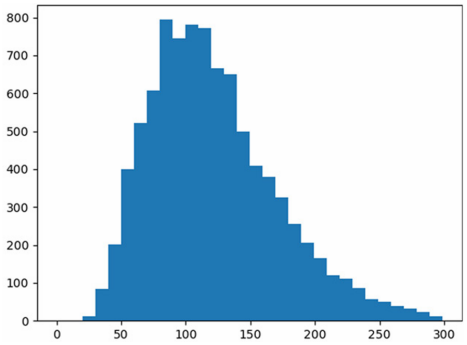


Figure 2.1: Rank tournament winner, 300 contestants, $0 < \sigma < 0.3$ (Aldous, 2019)

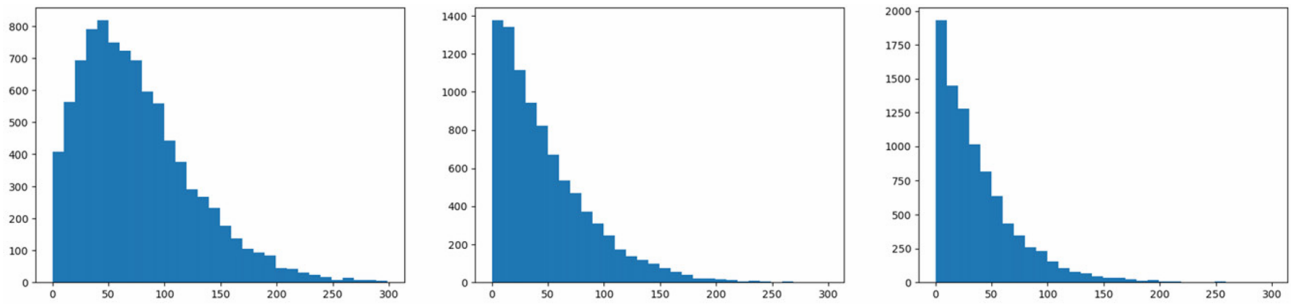


Figure 2.2: Rank tournament winner, 300 contestants, $0.05 < \sigma < 0.35$ (left), $0.1 < \sigma < 0.4$ (centre) and $0.15 < \sigma < 0.45$ (right) (Aldous, 2019)

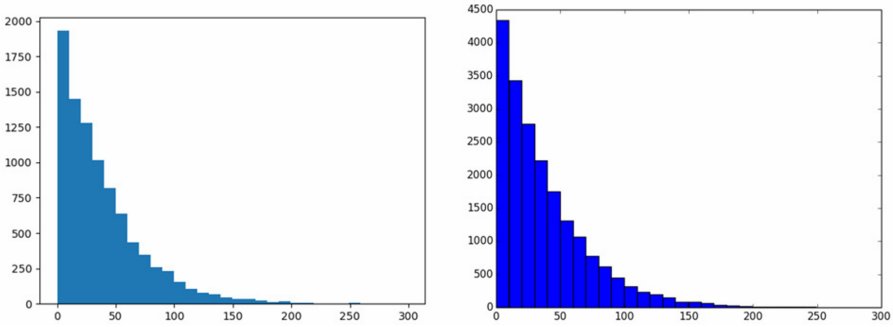


Figure 2.3: Ranks of tournament winner (left) and top 10 finishers (right), 300 contestants, $0.15 < \sigma < 0.35$ (Aldous, 2019)

3 Proper scoring rules

In Section 2, a score was used to evaluate how accurately a contestant predicted the true probability of a certain event occurring. This is known as a scoring rule. Definition 3.1 provides the formal definition of a scoring rule, and Definition 3.1 defines a proper scoring rule. The scoring rule used in the tournament in Section 2 is a proper scoring rule. The fact that the scoring rule is proper is important, as it ensures that the contestant whose predictions deviate least from the true probability will have the lowest expected score. In this section, we will also explore other examples of proper scoring rules.

Definition 3.1 (Scoring Rules)**.** Let $\mathcal{Y}$ be a set and $\mathcal{F}$ a σ-algebra on $\mathcal{Y}$. $\mathcal{Y}$ is considered the outcome space. When $\mathcal{Y} = \mathbb{R}^d$, we assume that $\mathcal{F}$ is the Borel σ-algebra. $\mathcal{P}$ denotes a convex set of probability measures on $(\mathcal{Y}, \mathcal{F})$. A scoring rule is a function

$$S : \mathcal{P} \times \mathcal{Y} \to \bar{\mathbb{R}},\ (\mathbb{P}, y) \to S(\mathbb{P}, y)$$

such that $S(\mathbb{P}, \mathbb{Q}) := \int S(\mathbb{P}, y)\mathrm{d}\mathbb{Q}(y)$ exists for all $\mathbb{P}, \mathbb{Q} \in \mathcal{P}$, but is not necessarily finite. (Waghmare & Ziegel, 2025)

Definition 3.2 (Proper Scoring Rules)**.** A scoring rule $S : \mathcal{P} \times \mathcal{Y} \to \bar{\mathbb{R}},\ (\mathbb{P}, y) \to S(\mathbb{P}, y)$, is called proper if

$$S(\mathbb{Q}, \mathbb{Q}) \leq S(\mathbb{P}, \mathbb{Q}) \tag{3.1}$$

for every $\mathbb{P}$, $\mathbb{Q} \in \mathcal{P}$ and strictly proper if (3.1) holds with equality if and only if $\mathbb{P} = \mathbb{Q}$. (Waghmare & Ziegel, 2025)

In other words, let y be an outcome of a random variable with distribution $\mathbb{Q}$. A scoring rule is a function that maps a predicted distribution $\mathbb{P}$, together with an outcome y, to a value S in $\bar{\mathbb{R}}$. The scoring rule is proper if $\mathbb{E}[S]$ is minimized when the predicted distribution $\mathbb{P}$ is equal to the true distribution $\mathbb{Q}$. Strictly proper scoring rules remain strictly proper under linear transformations (Toda, 1963).

Every proper scoring rule also induces a divergence $d(\mathbb{P}, \mathbb{Q}) = S(\mathbb{P}, \mathbb{Q}) - S(\mathbb{Q}, \mathbb{Q})$, which is nonnegative and equal to zero if and only if $\mathbb{P} = \mathbb{Q}$ (Buchweitz, Romano, & Tibshirani, 2025). This divergence quantifies how much the expected score under the predicted distribution $\mathbb{P}$ deviates from the expected score when the true distribution $\mathbb{Q}$ is predicted (which yields the lowest expected score). We call a scoring rule symmetric when the induced divergence d is symmetric, thus when $d(\mathbb{P}, \mathbb{Q}) = d(\mathbb{Q}, \mathbb{P})$. A symmetric scoring rule penalizes overestimation and underestimation of the true distribution equally (Buchweitz et al., 2025).

3.1 Brier Score

A well-known scoring rule, used in the tournament described in Section 2, is the Brier Score. The following example provides the definition of the Brier Score, originally proposed by Brier in 1950. The range of this score is $[0, 1]$.

Example 3.3 (Brier Score)**.** For a Binary outcome $y \in \{0, 1\}$, the Brier or quadratic score

$$S_{\text{Brier}}(\mathbb{P}, y) = (p - y)^2,$$

where $\mathbb{P}$ is the predicted distribution and p is the corresponding predicted value for the event $\{Y = 1\}$. The Brier score is a strictly proper scoring rule. (Brier, 1950)

In Equation 2.1, we showed that if p is the predicted probability and q is the true probability of an event occurring, then the expected score is:

$$S(\mathbb{P}, \mathbb{Q}) = q(1 - q) + (p - q)^2$$

We can use this to compute the divergence:

$$\begin{aligned} d(\mathbb{P}, \mathbb{Q}) &= S(\mathbb{P}, \mathbb{Q}) - S(\mathbb{Q}, \mathbb{Q}) \\ &= q(1 - q) + (p - q)^2 - q(1 - q) \\ &= (p - q)^2 \end{aligned} \tag{3.2}$$

We see that $d(\mathbb{P}, \mathbb{Q}) = d(\mathbb{Q}, \mathbb{P})$, so the Brier score is symmetric.

However, Brier's original definition of the score was intended for multi-category forecasts. For example, when predicting the probability of no rain, light rain, or heavy rain, the Multi-Category Brier Score can be used to assess a predictor.

Example 3.4 (Multi-Category Brier Score)**.** For an outcome $y \in \{0, 1, .., C\}$, the Brier Score is given by

$$S_{\text{Brier}}(\mathbb{P}, y) = \sum_{i=0}^{C} (p_i - y_i)^2,$$

where $\mathbb{P}$ is the predicted distribution and the p_i are the corresponding predicted values for the events $\{Y = i\}$. $y_i = 1$ if $y = i$, else $y_i = 0$. This is a strictly proper scoring rule. (Brier, 1950)

The expected score of the Multi-Category Brier score is

$$S(\mathbb{P}, \mathbb{Q}) = \sum_{j=0}^{C} q_j \left((1 - p_j)^2 + \sum_{i \neq j}^{C} p_i^2 \right) \tag{3.3}$$

This gives us the divergence:

$$\begin{aligned}
d(\mathbb{P}, \mathbb{Q}) &= S(\mathbb{P}, \mathbb{Q}) - S(\mathbb{Q}, \mathbb{Q}) \\
&= \sum_{j=0}^{C} q_j \left((1 - p_j)^2 + \sum_{i \neq j}^{C} p_i^2 \right) - \sum_{j=0}^{C} q_j \left((1 - q_j)^2 + \sum_{i \neq j}^{C} q_i^2 \right) \\
&= \sum_{j=0}^{C} q_j \left((1 - p_j)^2 - (1 - q_j)^2 + \sum_{i \neq j}^{C} p_i^2 - q_i^2 \right) \\
&= \sum_{j=0}^{C} q_j \left(1 - 2p_j + p_j^2 - 1 + 2q_j - q_j^2 + \sum_{i \neq j}^{C} p_i^2 - q_i^2 \right) \\
&= \sum_{j=0}^{C} q_j \left(2q_j - 2p_j + \sum_{i=0}^{C} p_i^2 - q_i^2 \right) \\
&= \sum_{j=0}^{C} 2q_j^2 - 2q_j p_j + \sum_{j=0}^{C} q_j \sum_{i=0}^{C} p_i^2 - q_i^2 \qquad (3.4) \\
&= \sum_{j=0}^{C} 2q_j^2 - 2q_j p_j + 1 \cdot \sum_{i=0}^{C} p_i^2 - q_i^2 \\
&= \sum_{j=0}^{C} 2q_j^2 - 2q_j p_j + p_j^2 - q_j^2 \\
&= \sum_{j=0}^{C} q_j^2 - 2q_j p_j + p_j^2 \\
&= \sum_{j=0}^{C} (q_j - p_j)^2
\end{aligned}$$

The divergence $d(\mathbb{P}, \mathbb{Q})$ for the Multi-Category Brier Score is defined as $\sum_i^C (p_i - q_i)^2$. Clearly, $d(\mathbb{P}, \mathbb{Q}) = d(\mathbb{Q}, \mathbb{P})$, so this scoring rule is symmetric as well.

We now move on to a generalized version of the Brier Score: the Power Score, introduced in Example 3.5. The Power Score includes a parameter β.

Example 3.5 (Power Score)**.** For $y \in \{1, ..., C\}$ and $\beta > 1$ the Power score is given by

$$S_{\text{power}}(\mathbb{P}, y) = -\beta p_y^{\beta-1} + (\beta - 1) \sum_i^C p_i^\beta$$

where $\mathbb{P}$ is the predicted distribution and the p_i are the corresponding predicted values for the events $\{Y = i\}$. This is a strictly proper scoring rule. (Selten, 1998)

When $\beta = 2$, the expression $\frac{1}{2}(1 + S_{\text{power}})$ is equal to the Brier Score. For the binary case the expectation of the Power score is:

$$S(\mathbb{P}, \mathbb{Q}) = \sum_j^C q_j \left(-\beta p_j^{\beta-1} + (\beta - 1) \sum_i^C p_i^\beta \right) \tag{3.5}$$

The divergence is:

$$\begin{aligned}
d(\mathbb{P}, \mathbb{Q}) &= S(\mathbb{P}, \mathbb{Q}) - S(\mathbb{Q}, \mathbb{Q}) \\
&= \sum_j^C q_j \left(-\beta p_j^{\beta-1} + (\beta - 1) \sum_i^C p_i^\beta \right) - \sum_j^C q_j \left(-\beta q_j^{\beta-1} + (\beta - 1) \sum_i^C q_i^\beta \right) \\
&= \sum_j^C q_j (\beta q_j^{\beta-1} - \beta p_j^{\beta-1}) + (\beta - 1) \sum_j^C q_j \sum_i^C p_i^\beta - q_i^\beta \\
&= \sum_j^C \beta q_j (q_j^{\beta-1} - p_j^{\beta-1}) + (\beta - 1) \cdot 1 \cdot \sum_i^C p_i^\beta - q_i^\beta \\
&= \sum_j^C \beta q_j^\beta - \beta q_j p_j^{\beta-1} + \beta p_j^\beta - \beta q_j^\beta - p_j^\beta + q_j^\beta \qquad (3.6) \\
&= \sum_j^C -\beta q_j p_j^{\beta-1} + \beta p_j^\beta - p_j^\beta + q_j^\beta \\
&= \sum_j^C \beta p_j^{\beta-1} (p_j - q_j) - (p_j^\beta - q_j^\beta)
\end{aligned}$$

(3.7)

Since $d(\mathbb{P}, \mathbb{Q}) \neq d(\mathbb{Q}, \mathbb{P})$, the Power Score, unlike the Brier Score, is not symmetric.

3.2 Logarithmic Score

Another common scoring rule is the Logarithmic Score. The Logarithmic Score has a close connection to the maximum likelihood principle and has a range of $[0, \infty]$. It attains a value of ∞ when the predictor assigns a probability that is impossible to an event. For example, if a predictor assigns a probability of 0 to the event $Y = 1$, and the observed outcome is $y = 1$, the predictor receives a score of ∞. Even if the true probability of the event $Y = 1$ is very small, the Logarithmic Score effectively makes predicting a probability of 0 unforgivable.

It is also hypersensitive to small differences in small probabilities, the expected score reacts strongly to small changes in low probability predictions, while being insufficiently sensitive for higher probabilities. These characteristics can make the Logarithmic Score somewhat undesirable in practice (Selten, 1998).

When there are more than two possible outcomes, the Logarithmic Score is the only proper scoring rule whose value depends only on the predicted probability of the correct outcome. This property is also called the local property (Shuford, Albert, & Massengill, 1966).

Example 3.6 (Logarithmic Score)**.** The logarithmic score is given by

$$S_{\log}(\mathbb{P}, y) = -\log(p(y)),$$

where p is the density or the probability mass function of $\mathbb{P}$ and $\log(0)$ is defined as ∞. This is a strictly proper scoring rule. (Good, 1952)

For the binary case the expectation of the logarithmic score is:

$$S(\mathbb{P}, \mathbb{Q}) = -q \log(p) - (1-q) \log(1-p) \tag{3.8}$$

The divergence is:

$$\begin{aligned} d(\mathbb{P}, \mathbb{Q}) &= S(\mathbb{P}, \mathbb{Q}) - S(\mathbb{Q}, \mathbb{Q}) \\ &= -q \log(p) - (1-q) \log(1-p) + q \log(q) + (1-q) \log(1-q) \\ &= q \log\left(\frac{q}{p}\right) + (1-q) \log\left(\frac{1-q}{1-p}\right) \end{aligned} \tag{3.9}$$

$d(\mathbb{P}, \mathbb{Q}) \neq d(\mathbb{Q}, \mathbb{P})$, so the Logarithmic score is not symmetric.

3.3 Spherical Score

Together with the Brier and Logarithmic score, the Spherical score is one of the most popular and well studied scoring rule. The Spherical score was first introduced in the context of psychological testing (Jose, 2007).

Example 3.7 (Spherical Score)**.** For $y \in \{0, 1, ..., C\}$ the spherical score is given by

$$S_{\text{spherical}}(\mathbb{P}, y) = -\frac{p_y}{\sqrt{p_1^2 + p_2^2 + ... + p_C^2}}$$

where $\mathbb{P}$ is the predicted distribution and the p_i are the corresponding predicted values for the events $\{Y = i\}$. This is a strictly proper scoring rule.(Roby, 1965)

The Spherical score has a range of $[-1, 0]$. In the binary case $y \in \{0, 1\}$, the expected score is:

$$S(\mathbb{P}, \mathbb{Q}) = -\frac{p}{\sqrt{p^2 + (1-p)^2}} q - \frac{1-p}{\sqrt{p^2 + (1-p)^2}} (1-q) = -\frac{pq + (1-p)(1-q)}{\sqrt{p^2 + (1-p)^2}} \tag{3.10}$$

The resulting divergence is:

$$\begin{aligned} d(\mathbb{P}, \mathbb{Q}) &= S(\mathbb{P}, \mathbb{Q}) - S(\mathbb{Q}, \mathbb{Q}) \\ &= -\frac{pq + (1-p)(1-q)}{\sqrt{p^2 + (1-p)^2}} + \frac{q^2 + (1-q)^2}{\sqrt{q^2 + (1-q)^2}} \\ &= -\frac{pq + (1-p)(1-q)}{\sqrt{p^2 + (1-p)^2}} + \sqrt{q^2 + (1-q)^2} \end{aligned} \tag{3.11}$$

$d(\mathbb{P}, \mathbb{Q}) \neq d(\mathbb{Q}, \mathbb{P})$, so like the Power score and Logarithmic score, the Spherical score is not symmetric.

A variation of the Spherical is the Pseudospherical Score with parameter β in Example 3.8. For $\beta = 2$, the Pseudospherical score is equal to the Spherical score.

Example 3.8 (Pseudospherical Score)**.** For $y \in \{0, ..., C\}$ and $\beta > 1$ the Pseudospherical score is given by

$$S_{\text{spherical}}(\mathbb{P}, y) = -\left(\frac{p_y}{\sqrt[\beta]{p_1^\beta + p_2^\beta + ... + p_C^\beta}}\right)^{\beta - 1}$$

where p_i is the predicted value for the event $\{Y = i\}$. This is a strictly proper scoring rule. (Selten, 1998)

For the binary case $y \in \{0, 1\}$, the expected score is:

$$S(\mathbb{P}, \mathbb{Q}) = -\frac{p^{\beta-1}}{\sqrt[\beta]{p^\beta + (1-p)^{\beta^{\beta-1}}}} q - \frac{(1-p)^{\beta-1}}{\sqrt[\beta]{p^\beta + (1-p)^{\beta^{\beta-1}}}} (1-q) = -\frac{p^{\beta-1} q + (1-p)^{\beta-1} (1-q)}{\sqrt[\beta]{p^\beta + (1-p)^{\beta^{\beta-1}}}} \tag{3.12}$$

The resulting divergence is:

$$\begin{aligned} d(\mathbb{P},\mathbb{Q}) &= S(\mathbb{P},\mathbb{Q}) - S(\mathbb{Q},\mathbb{Q}) \\ &= -\frac{p^{\beta-1}q + (1-p)^{\beta-1}(1-q)}{\sqrt[\beta]{p^\beta + (1-p)^\beta}^{\beta-1}} + \frac{q^\beta + (1-q)^\beta}{\sqrt[\beta]{q^\beta + (1-q)^\beta}^{\beta-1}} \\ &= -\frac{p^{\beta-1}q + (1-p)^{\beta-1}(1-q)}{\sqrt[\beta]{p^\beta + (1-p)^\beta}^{\beta-1}} + (q^\beta + (1-q)^\beta)^{1-\frac{\beta-1}{\beta}} \\ &= -\frac{p^{\beta-1}q + (1-p)^{\beta-1}(1-q)}{\sqrt[\beta]{p^\beta + (1-p)^\beta}^{\beta-1}} + (q^\beta + (1-q)^\beta)^{\frac{1}{\beta}} \end{aligned} \tag{3.13}$$

As we saw with the Spherical Score, $d(\mathbb{P},\mathbb{Q}) \neq d(\mathbb{Q},\mathbb{P})$, so the Pseudospherical Score is not symmetric.

3.4 Continuous Ranked Probability Score

Another scoring rule we would like to discuss is the Continuous ranked probability score in Example 3.9. It can be interpreted as the Brier score for the event $\mathbf{1}_{\{Y\leq x\}}$ integrated over $\mathbb{R}$ and is often used in meteorology (Waghmare & Ziegel, 2025).

Example 3.9 (Continuous ranked probability score (CRPS))**.** For a real-valued outcome $y \in \mathbb{R}$, the continuous ranked probability score (CRPS) is defined as

$$CRPS(\mathbb{P}, y) = \int (F_{\mathbb{P}}(x) - \mathbf{1}\{y \leq x\})^2 \, \mathrm{d}x,$$

where $F_{\mathbb{P}}$ is the cumulative distribution function (CDF) of $\mathbb{P}$. This is a strictly proper scoring rule. (Matheson & Winkler, 1976)

In the case where y is drawn from a Bernoulli(p) random variable and $F_{\mathbb{P}}$ is the cumulative distribution function of a Bernoulli(q) random variable, the Continuous Ranked Probability Score is equal to the Brier Score.

When dealing with a multi-categorical random variable, where $y \in 0, \ldots, C$, and a contestant predicts probabilities $p_0, p_1, \ldots, p_C$ for each category (with the constraint that $\sum_{i=0}^{C} p_i = 1$), the CRPS simplifies to:

$$\begin{aligned} CRPS(\mathbb{P}, y) &= \int_{-\infty}^{\infty} (F_{\mathbb{P}}(x) - \mathbf{1}\{y \leq x\})^2 \, \mathrm{d}x \\ &= \int_{-\infty}^{y^-} (F_{\mathbb{P}}(x) - \mathbf{1}\{y \leq x\})^2 \, \mathrm{d}x + \int_{y}^{\infty} (F_{\mathbb{P}}(x) - \mathbf{1}\{y \leq x\})^2 \, \mathrm{d}x \\ &= \int_{-\infty}^{y^-} F_{\mathbb{P}}(x)^2 \, \mathrm{d}x + \int_{y}^{\infty} (F_{\mathbb{P}}(x) - 1)^2 \, \mathrm{d}x \\ &= \sum_{i=0}^{y-1} p_i^2 \, \mathrm{d}x + \sum_{i=y}^{C-1} (p_i - 1)^2 \, \mathrm{d}x \end{aligned} \tag{3.14}$$

The expectation of the Continuous ranked probability score is:

$$\begin{aligned} S(\mathbb{P},\mathbb{Q}) &= \int_{-\infty}^{\infty} f_{\mathbb{Q}}(y) \int_{-\infty}^{\infty} (F_{\mathbb{P}}(x) - \mathbf{1}\{y \leq x\})^2 \, \mathrm{d}x \, \mathrm{d}y \\ &= \int_{-\infty}^{\infty} f_{\mathbb{Q}}(y) \int_{-\infty}^{\infty} F_{\mathbb{P}}(x)^2 - 2F_{\mathbb{P}}(x)\mathbf{1}\{y \leq x\} + \mathbf{1}\{y \leq x\}^2 \, \mathrm{d}x \, \mathrm{d}y \\ &= \int_{-\infty}^{\infty} f_{\mathbb{Q}}(y) \, \mathrm{d}y \int_{-\infty}^{\infty} F_{\mathbb{P}}(x)^2 \, \mathrm{d}x - 2\int_{-\infty}^{\infty} F_{\mathbb{P}}(x) \int_{-\infty}^{x} f_{\mathbb{Q}}(y) \, \mathrm{d}y \, \mathrm{d}x + \int_{-\infty}^{\infty}\int_{-\infty}^{x} f_{\mathbb{Q}}(y) \, \mathrm{d}y \, \mathrm{d}x \\ &= \int_{-\infty}^{\infty} F_{\mathbb{P}}(x)^2 \, \mathrm{d}x - 2\int_{-\infty}^{\infty} F_{\mathbb{P}}(x)F_{\mathbb{Q}}(x) \, \mathrm{d}x + \int_{-\infty}^{\infty} F_{\mathbb{Q}}(x) \, \mathrm{d}x \end{aligned} \tag{3.15}$$

The divergence is:

$$\begin{aligned} d(\mathbb{P},\mathbb{Q}) &= S(\mathbb{P},\mathbb{Q}) - S(\mathbb{Q},\mathbb{Q}) \\ &= \int_{-\infty}^{\infty} F_{\mathbb{P}}(x)^2 - 2F_{\mathbb{P}}(x)F_{\mathbb{Q}}(x) + F_{\mathbb{Q}}(x)\ \mathrm{d}x\ - \int_{-\infty}^{\infty} F_{\mathbb{Q}}(x)^2 - 2F_{\mathbb{Q}}(x)^2 + F_{\mathbb{Q}}(x)\ \mathrm{d}x \\ &= \int_{-\infty}^{\infty} F_{\mathbb{P}}(x)^2 - 2F_{\mathbb{P}}(x)F_{\mathbb{Q}}(x) + F_{\mathbb{Q}}(x)^2 \mathrm{d}x \\ &= \int_{-\infty}^{\infty} (F_{\mathbb{P}}(x) - F_{\mathbb{Q}}(x))^2 \mathrm{d}x \end{aligned} \tag{3.16}$$

$d(\mathbb{P},\mathbb{Q}) = d(\mathbb{Q},\mathbb{P})$, so the Continuous ranked probability score is symmetric.

4 Further analyis of the original model

4.1 Finish positions

Beyond examining just the rank of the winner, it is also insightful to consider the average finish position of each contestant. In Figure 2.1, we saw that the 100th most accurate contestant was most likely to win the tournament in the case where σ ranges from 0 to 0.3. If the 100th most accurate contestant occasionally achieves a top score due to luck, then they must also perform poorly in other tournaments due to bad luck. Otherwise, this would contradict the fact that the Brier score is a proper scoring rule. Therefore, it does not necessarily follow that the 100th most accurate contestant has the lowest average finish position. At the same time, if the most accurate predictors rarely win the tournament, it is natural to ask: where do they typically finish?

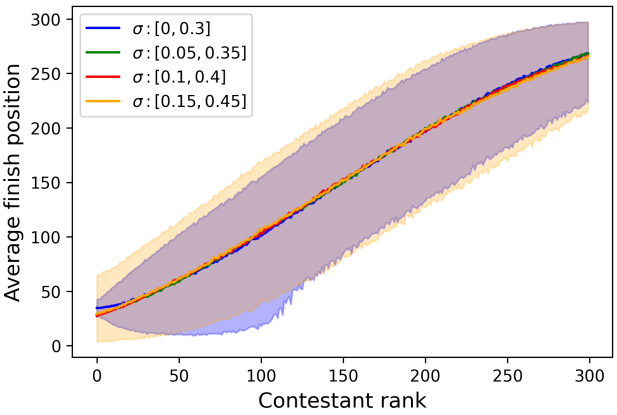


Figure 4.1: Average finish position per contestant for different σ ranges

Figure 4.1 shows the average finish position per contestant for σ in ranges [0, 0.3] (blue), [0.05, 0.35] (green), [0.1, 0.4] (red) and [0.15, 0.45] (yellow). The blue-colored area represents the region between the 0.1 and 0.9 quantiles for σ in the range [0, 0.3]. The yellow-colored area shows the same quantile range, but for σ in the range [0.15, 0.45]. These two regions do overlap in Figure 4.1 and create a mixed color. For clarity, the region between the 0.1 and 0.9 quantiles for σ in ranges [0.05, 0.35] and [0.1, 0.4] is omitted in Figure 4.1.

We can observe that, for σ between 0 and 0.3, the contestant with the lowest average finish position is not the 100th most accurate contestant, but actually the most accurate one. For all intervals of σ, the average finish position increases strictly with contestant rank. However, we do observe that the average finish position of the most accurate contestant for σ between 0 and 0.3 is 35. This is slightly higher than the average finish position of around 28 for σ in the other intervals.

The space between the 0.1 and 0.9 quantiles is slightly wider for σ in the interval [0.15, 0.45] than for σ in [0, 0.3]. This space is particularly narrow around the most accurate contestants when σ is in [0, 0.3]. Another notable difference is that for σ in [0.15, 0.45], the quantiles increase along with the average. In contrast, for σ in [0, 0.3], the lower quantile actually decreases up to approximately the 100th-ranked contestant before increasing again. This behavior is consistent with the findings in Figure 2.1 and Figure 2.2. As shown in Figure 2.1, contestants ranked around 100 are the most likely to win the tournament, so we expect them to achieve lower ranks more frequently. On the other hand, for $0.15 < \sigma < 0.45$ in Figure 2.2, the most accurate contestant is most likely to win, so we do not expect a decrease in the lower quantile.

In Figure 4.1, we saw that the average finish position of the most accurate contestant for $0 < \sigma < 0.3$ was 35. It may be interesting to take a closer look at the distribution of this contestant's finish positions. Figure 4.2 shows this distribution. We can see that the most accurate contestant is most likely to finish around position 35 and generally does not place lower than 20 or higher than 50. This indicates a relatively narrow range of possible finish positions, which is also reflected in Figure 4.1 by the small gap between the 0.1 and 0.9 quantiles.

We may also want to consider the reverse of Figure 4.1. Instead of showing the average finish position per contestant, Figure 4.3 presents the average contestant rank per finish position for σ in the intervals [0, 0.3] (blue), [0.05, 0.35] (green), [0.1, 0.4] (red), and [0.15, 0.45] (yellow). For σ in the first interval, [0, 0.3], the winner (finish position 0) has a high average rank of 124. This value decreases significantly across the higher intervals: to 75 for [0.05, 0.35], 48 for [0.1, 0.4], and 36 for [0.15, 0.45]. These findings align with Figures 2.1 and 2.2, where increasing σ intervals result in more accurate (lower-ranked) contestants being more likely to win.

For σ between 0 and 0.3, we observe a noticeable dip in average rank for the lower finish positions. The lowest average rank corresponds to finish position 35. For the other intervals, the finish positions with the lowest average ranks are 14, 8, and 2, respectively. As σ increases, this dip diminishes, and the relationship between finish position and average rank turns into a function increasing over more of its domain.

All of this suggests that, in the case of σ between 0 and 0.3, it may be more informative to focus on finish

position 35 rather than the tournament winner. Figure 4.4 illustrates this. We observe that the most accurate contestants are indeed the most likely to finish 35th, while contestants ranked higher than 10 finish in this position significantly less frequently.

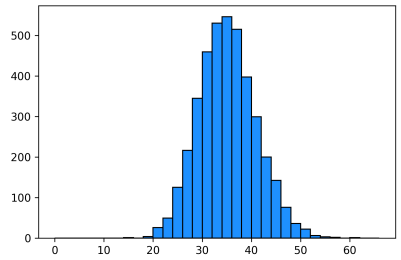


Figure 4.2: Finish positions most accurate contestant, 300 contestants, $0 < \sigma < 0.3$

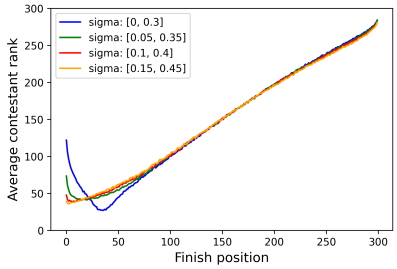


Figure 4.3: Average rank for each finish position, 300 contestants, $0 < \sigma < 0.3$

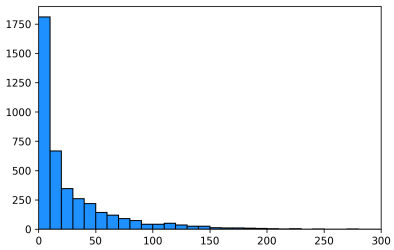


Figure 4.4: Ranks of the 35th finish position, 300 contestants, $0 < \sigma < 0.3$

It appears that we can still identify the most accurate contestant fairly reliably by looking at the 35th-place finisher when σ is between 0 and 0.3. However, does this method of bypassing the paradox hold for other values of σ? To investigate this, we analyze the case where σ lies between 0.05 and 0.35. Figure 4.5 shows the contestant ranks corresponding to finish position 14, as this position had the lowest average rank in that σ range. We observe some improvement in Figure 4.5 compared to Figure 2.2: more accurate contestants appear more frequently at the selected finish position. However, the improvement is not nearly as pronounced as in Figure 4.4. This is because the minimum average contestant rank for σ between 0.05 and 0.35 is significantly higher than for σ between 0 and 0.3, as shown in Figure 4.3. This difference can be attributed to the increased variability in the finish positions of the most accurate contestant. As illustrated in Figure 4.6, the most accurate contestant finishes between positions 0 and 80 for σ in the range [0.05, 0.35], which is a much wider range than for [0, 0.3] in Figure 4.2 . Additionally, the distribution of these finish positions is more spread out, whereas in the [0, 0.3] range, the distribution is more sharply peaked.

So, for σ in the range [0.05, 0.35], the most accurate contestant does not consistently finish in a single position, such as 14th, as frequently. This leaves more room for less accurate contestants to also finish in that position. As a result, identifying the most accurate contestant by looking at the 14th-place finisher is less effective than using the 35th-place finisher in the case where σ is between 0 and 0.3.

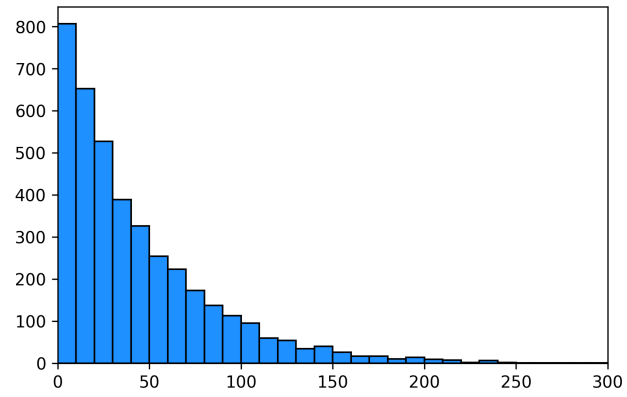


Figure 4.5: Ranks of the 14th finish position, 300 contestants, $0.05 < \sigma < 0.35$

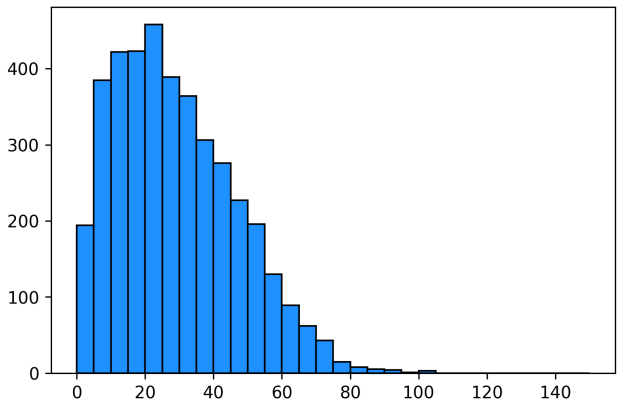


Figure 4.6: Finish positions most accurate contestant, 300 contestants, $0.05 < \sigma < 0.35$

4.2 Scores

In Section 4.1, we analyzed the finishing positions of contestants. Another point of interest is the contestants' scores themselves. Since the effect of the paradox is most pronounced when σ lies between 0 and 0.3 (see Section

2 for further details), this section will primarily focus on this scenario.

We begin by examining the average score of each contestant. Figure 4.7 displays these averages (blue line), the region between the 0.1 and 0.9 quantiles (dark blue), and the full range of scores (light blue) for each contestant. As expected, the average score increases with contestant rank. This is because the Brier score is a proper scoring rule, meaning the most accurate predictor has the lowest expected score.

The 0.1 and 0.9 quantiles also rise with increasing rank, and while the spread between them grows, the increase in spread is relatively small. This spread is a result of the variance in score increasing as σ increases, which also contributes to the broader ranges visible in Figure 4.7.

However, the lowest score achieved by each contestant does not increase as smoothly as the average or the 0.1 quantile. Up to around rank 100, there's little to no visible upward trend. Beyond rank 100, we still observe contestants achieving very low scores, sometimes even lower than those of the most accurate predictor (rank 0). While one might dismiss these as outliers, they could significantly impact the tournament outcome as we are primarily concerned with identifying the winner. Scores higher than the lowest score are not taken into consideration.

Figure 4.8 presents boxplots of the scores for contestants ranked 0, 50, 100, 150, and 200. Similar conclusions can be drawn from these boxplots as from the results shown in Figure 4.7. The interquartile range increases with rank, which can be attributed to increasing variance. As with the average, the median score also rises with rank. For all contestants, there are more outliers towards the higher scores than the lower scores. However, for contestants ranked 100 and above, we observe more low-score outliers compared to those ranked 0 and 50. Notably, for rank 100, the lowest outliers are roughly at the same level as those for ranks 0 and 50. This suggests that contestants ranked around 100 can occasionally achieve scores that outperform those of the most accurate predictor (rank 0) in a tournament setting.

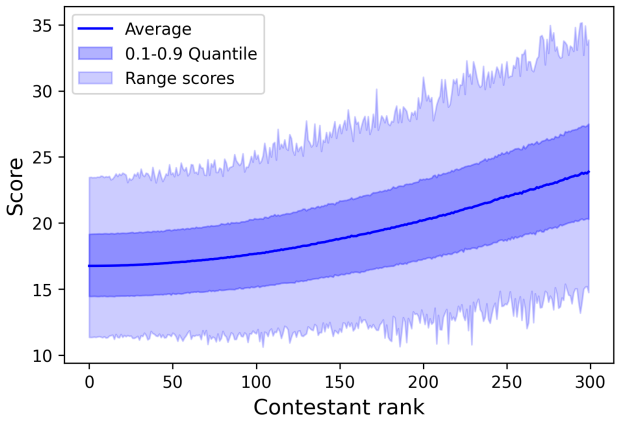


Figure 4.7: Average, 0.1-0.9 quantile and range of scores per contestant, $0 < \sigma < 0.3$

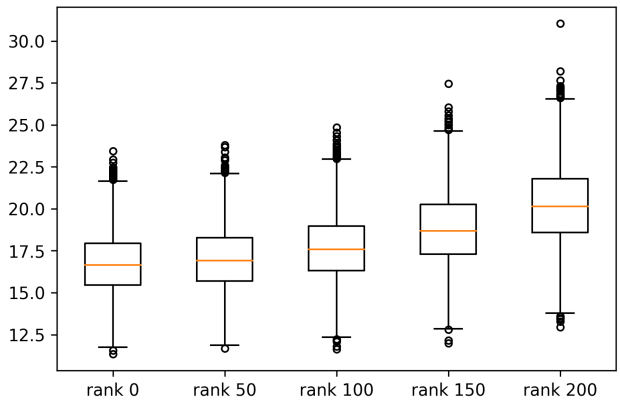


Figure 4.8: Boxplots of the scores for contestant ranked 0, 50, 100, 150 and 200, $0 < \sigma < 0.3$

In Figures 4.7 and 4.8, we compared the scores of the most accurate contestant to those of contestants with different ranks. But how do the scores of the most accurate contestant compare to those of the actual tournament winner? Figure 4.9 shows a scatter plot with on the horizontal axis the score of the winner and on the vertical axis the score of the most accurate predictor. The black line is where their scores would be the same, which is the best possible outcome. We cannot have any points under the black line as then the score of the best predictor would be lower than the score of the winner, which is not possible as then the best predictor would be the winner. We can see that the points lay a slightly above this line, but never touch it. This is because, as shown in Figure 2.1, the most accurate contestant never wins the tournament, so there is always a slight difference in score between the winner and the most accurate predictor. In Figure 4.9 we can also see that the scores of the winner and best predictor are highly correlated. Lower scores for the winner, result in lower scores for the best predictor. This is because the scores depend on the outcomes of the same Bernoulli random variables. If the outcomes of the random variables are 'unexpected' (1 if the true success probability p is low or 0 if p is high), this will result in higher scores for all contestants. The opposite is also true. If the outcomes of the random variables are as 'expected' (1 if the true success probability p is high or 0 if p is low), the scores of all contestants will be lower.

To get a better idea of how often each score is achieved, Figure 4.10 presents a histogram of the scores of the most accurate contestant (red) alongside the scores of the winner of each tournament (blue). The two distributions are similar, but the scores of the most accurate predictor appear to be shifted approximately one point to the right. This shift is expected as otherwise the best predictor would achieve lower scores than the winner of the tournament. This is not possible. At best, the winner and the most accurate predictor can achieve the same scores. The peak of the scores of the best predictor is also slightly lower than the peak of the winner.

In addition to examining the score distributions of the tournament winner and the most accurate predictor, we also analyze the distribution of the differences in score between them. Figure 4.11 shows a histogram of the difference in score between the tournament winner and the most accurate contestant. As noted in Figure 4.10, the score distribution of the best predictor appears to be shifted approximately one point to the right compared to that of the winner. Consistent with this observation, Figure 4.11 shows that score differences around 1 are the most common. However, differences greater than 2 are not uncommon, and differences exceeding 3 are also observed.

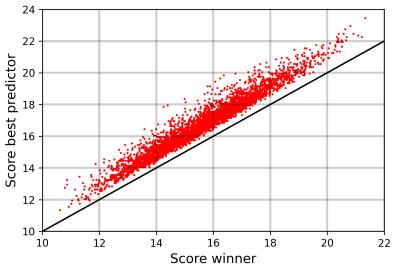


Figure 4.9: Scatter plot scores winner compared to the most accurate predictor, $0 < \sigma < 0.3$

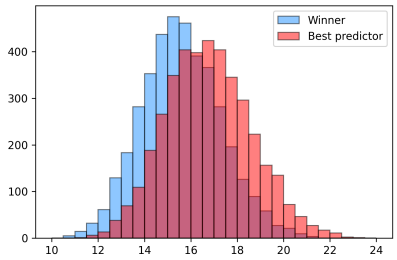


Figure 4.10: Scores winner (blue) compared to the most accurate predictor (red), $0 < \sigma < 0.3$

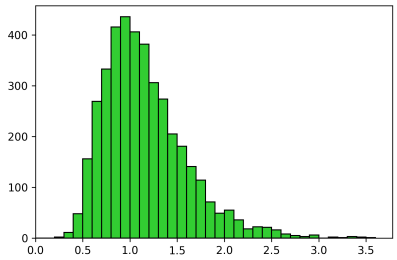


Figure 4.11: Differences score winner vs most accurate predictor, $0 < \sigma < 0.3$

One might also wonder how often each contestant outperforms the most accurate predictor. Figure 4.12 presents these results. On average, the most accurate predictor is beaten by 34.6 contestants per tournament. Lower ranked contestants beat the most accurate predictor more frequently than higher ranked ones. Contestants ranked around 300 almost never outperform the best predictor in terms of score.

Contestants ranked between 1 and 20 are approximately four times more likely to beat the most accurate predictor than contestants ranked around 100. So why, then, are contestants ranked around 100 winning the tournament much more frequently than others? The reason lies in the fact that we are not just interested in beating the most accurate predictor, but in outperforming all other contestants. Therefore, the magnitude by which a contestant beats the most accurate predictor also matters.

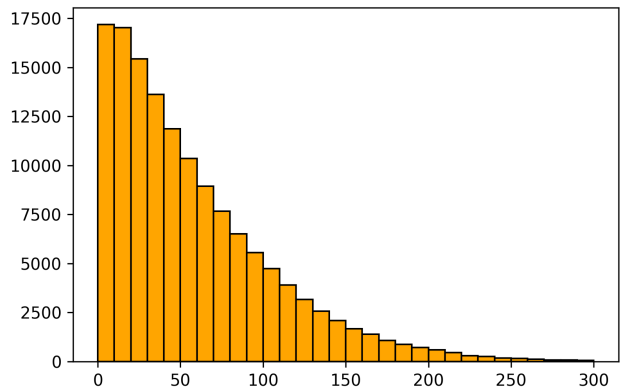


Figure 4.12: How often each ranked contestant beats the most accurate contestant in score, $0 < \sigma < 0.3$

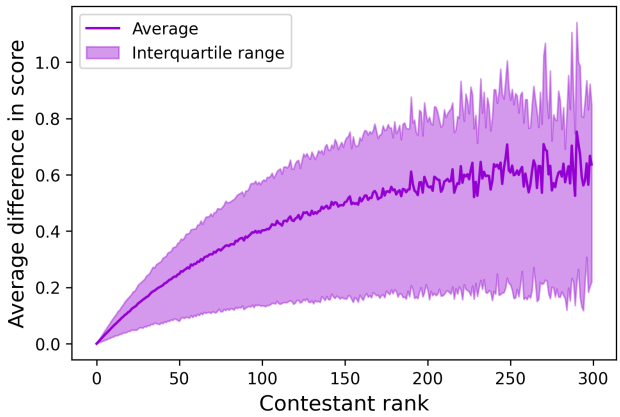


Figure 4.13: Average score difference when beating the most accurate contestant, $0 < \sigma < 0.3$

Figure 4.13 shows the average difference between the score of the most accurate predictor and the score of

a contestant who beats the most accurate predictor. The light purple area represents the interquartile range. To generate this figure we simulated the 50,000 tournaments instead of 4000. We did this to get a smoother and more accurate average for the contestants with higher ranks, who do not beat the most accurate contestant as often. As we can see, both the average score difference and the interquartile range increase with contestant rank. For contestants ranked between 200 and 300, these values become more erratic, as these contestants do not beat the most accurate predictor often enough to produce a smooth average. Overall, the average score difference increases more rapidly for lower ranked contestants and then flattens out as the rank increases.

From Figures 4.12 and 4.13, we can conclude that higher ranked contestants do not beat the most accurate predictor often, but when they do, they do so by a much larger margin than lower ranked contestants. According to Figure 2.1, around rank 100 appears to be the sweet spot, where contestants manage to beat the most accurate predictor often enough and by a large enough margin to win the tournament.

4.3 Top n

In Figure 2.3, the ranks of the top 10 finishers are compared to the rank of the tournament winner for σ between 0.15 and 0.35. D. J. Aldous concluded that the paradoxical effect applies similarly to the top 10 finishers as it does to the winner (Aldous, 2019). Since the effect of the paradox is strongest when σ is between 0 and 0.3, it is interesting to examine whether the same pattern holds for the top finishers in that range.

Figure 4.14 displays the ranks of the top 5, 10, and 20 finishers for $0 < \sigma < 0.3$. We observe that each distribution has a similar shape to that in Figure 2.1 (which shows the rank of the winner), but the peak is shifted to the left. This leftward shift becomes more pronounced as the number of top finishers increases from 5 to 10 to 20.

Notably, the 10 most accurate contestants almost never appear in the top 5 and only rarely make it into the top 10 or top 20 finishers.

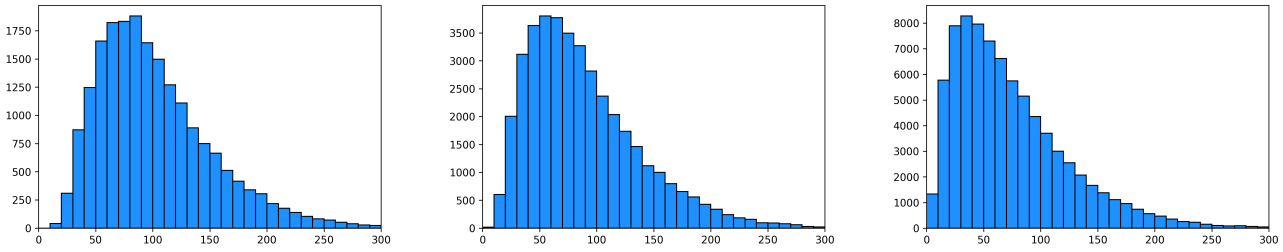


Figure 4.14: Ranks of the top 5, 10 and 20 finishers, prediction tournament with 300 contestants, $0 < \sigma < 0.3$

4.4 Conclusions

In Section 4.1, we examined the average finish position for each contestant rank. We observed that the average finish position increased with contestant rank for σ in the ranges [0, 0.3], [0.05, 0.35], [0.1, 0.4], and [0.15, 0.45]. For σ between 0 and 0.3, the most accurate predictor is most likely to finish 35th. We also analyzed the average rank per finish position. For the same range of σ, [0, 0.3], we found that the lowest average rank is achieved at finish position 35. When focusing on the ranks at finish position 35, rather than at the first finish position, we observed significantly better results compared to Figure 2.1. Contestants outside the top 10 most accurate contestants finish 35th far less frequently than those within the top 10. Applying the same technique for σ between 0.05 and 0.35 also yielded improved results, though the improvement was less significant.

Section 4.2 discussed the scores of contestants for σ between 0 and 0.3. The average score increases as contestant rank increases, and the range of scores achieved also widens with rank. We compared the scores of the tournament winner to those of the most accurate predictor. Both have similar score distributions, although the distribution for the most accurate predictor is shifted a full point to the right compared to that of the winner. When examining the distribution of score differences between the winner and the best predictor, we found that a difference of 1 point is the most common. Additionally, we observed that higher-ranked contestants do not outperform the most accurate predictor very often, but when they do, they tend to win by a significantly larger margin than lower-ranked contestants.

Section 4.3 analyzed the top 5, 10, and 20 finishers for $0 < \sigma < 0.3$. The distribution of ranks is similar to that of the tournament winner but the mode shifts progressively to the left as the number of top finishers increases. The 10 most accurate contestants never appear in the top 5 and rarely appear in the top 10 or top 20.

5 Distribution of the success probabilities

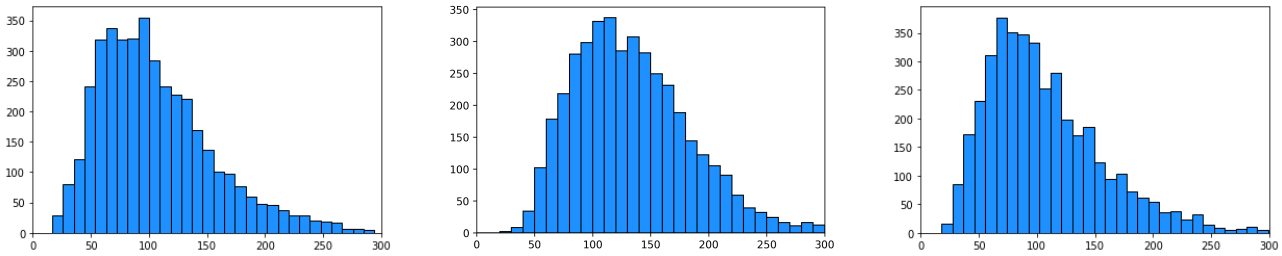
In the tournament described in Section 2, we considered 100 events with success probabilities $0.05, 0.15, 0.25, ..., 0.95$, each occurring 10 times. We now investigate the effect of modifying the distribution of these event probabilities. Specifically, we ask whether the effects described in Section 2 still persist, or whether they are influenced by changes in the distribution of event probabilities.

5.1 Shorter intervals

Instead of evenly distributing the event probabilities over the interval [0,1], we now explore what happens when the probabilities are concentrated around 0, 0.5, or 1. To do this, we define three distinct scenarios:

1. **Left:** The success probabilities of events are $0.05, 0.10, 0.15, 0.20$ and 0.25, where each probability occurs 20 times
2. **Middle:** The success probabilities of events are $0.40, 0.45, 0.50, 0.55$ and 0.60, where each probabilities occurs 20 times
3. **Right:** The success probabilities of events are $0.75, 0.80, 0.85, 0.90$ and 0.95, where each probability occurs 20 times

In each scenario, the total number of events remains 100. Scenario 1 focuses on probabilities skewed toward the lower end of the interval [0,1], scenario 2 on the central region, and scenario 3 on the upper end. Throughout, we maintain the same range of [0, 0.3] for σ.



Scenario 1:
$p \in \{0.05, 0.10, 0.15, 0.20, 0.25\}$

Scenario 2:
$p \in \{0.40, 0.45, 0.50, 0.55, 0.60\}$

Scenario 3:
$p \in \{0.75, 0.80, 0.85, 0.90, 0.95\}$

Figure 5.1: Rank tournament winner, probabilities in the left, middle or right side of the interval [0,1] for $0 < \sigma < 0.3$

Figure 5.1 shows the results of these different scenarios. We see similar outcomes in scenario 1 and scenario 3: in both cases, the tournament winner is most likely to come from around the 75th most accurate predictor. These histograms differ notably from the one in Figure 2.1, where the winner is most likely to come from around the 100th most accurate predictor. In the left and right scenarios, the distributions are shifted more to the left, indicating that more accurate contestants are more likely to win the tournament.

The similar results for scenario 1 and 3 are a result of the symmetric setup of the tournament. The truncation is applied symmetrically on both sides of the interval and the predicted probability for each contestant is $q = p \pm \sigma$. We also saw in Section 3.1 that the Brier score is symmetric, so it does penalizes over and underestimating the true probability by the same margin. These facts combined ensure that choosing event probabilities closer to the right side of the interval [0,1] should have the same effect as choosing them closer to the left side.

However, scenario 2 looks different from scenario 1 and 3. The ranks of the tournament winners are shifted more to the right, compared to the ranks in Figure 2.1. In this case, the winner is most likely to come from around the 115th most accurate contestant.

The difference between scenarios 1/3 and scenario 2 can be explained by the effect of constraining the predicted probability q to the interval [0, 1]. When the true probabilities p are closer to 0 or 1, the values of $p \pm \sigma$ are more likely to be truncated to 0 or 1 for larger values of σ. This truncation affects contestant performance: when a prediction of $p + \sigma$ (if the outcome is 1) or $p - \sigma$ (if the outcome is 0) is truncated, the reward, defined as the reduction in score relative to predicting p, is diminished compared to cases where no

truncation occurs. Furthermore, if a contestant predicts in the wrong direction ($p - \sigma$ when the event occurs, or $p + \sigma$ when it does not), they are penalized just as much as they would be without truncation. As a result, contestants with larger σ, who experience more frequent truncation in scenarios 1 and 3, perform relatively worse compared to when using the original distribution of probabilities. In contrast, in scenario 2, truncation to 0 or 1 does not occur for any contestant, since their predictions deviate by at most 0.3 from p, and p lies within the interval [0.4, 0.6]. This has the opposite effect: contestants with larger σ perform relatively better compared to when the original probability distribution is used.

We are also interested in examining what happens when the probabilities are shifted more toward the left and right ends of the interval [0,1], eliminating events with probabilities near 0.5. To achieve this, we use the following distribution of probabilities: events have probabilities 0.05, 0.10, 0.15, 0.20, 0.25, 0.75, 0.80, 0.85, 0.90, and 0.95, with each probability occurring 10 times. This setup still results in 100 events. We also maintain σ between 0 and 0.3.

Figure 5.2 presents the results for this scenario. We observe that the outcomes are very similar to those shown in scenarios 1 and 3 in Figure 5.1. It appears that the specific side of the interval [0, 1] toward which the probabilities are skewed does not significantly affect the results. This is because both the predictions made by the contestants and the Brier scoring rule are symmetric. What truly matters is the distance of the probabilities from the center of the interval, rather than the direction of the shift.

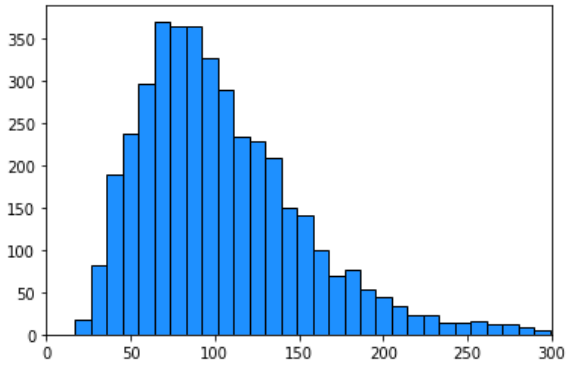


Figure 5.2: Rank tournament winner, probabilities in the left and right side of the interval [0,1]

5.2 Beta distribution

While the results in Section 5.1 are interesting, in real-life scenarios, the probabilities are unlikely to be this evenly distributed or exactly the same every time. Therefore, we want to explore the case where these probabilities are sampled from a probability distribution. Since the Beta distribution is restricted to the interval [0, 1] and is highly versatile, it is well suited for modeling success probabilities. We use the following definition of the Beta distribution.

Definition 5.1 (Beta Distribution)**.** The beta distribution has parameters $\alpha > 0$ and $\beta > 0$ and probability density function:

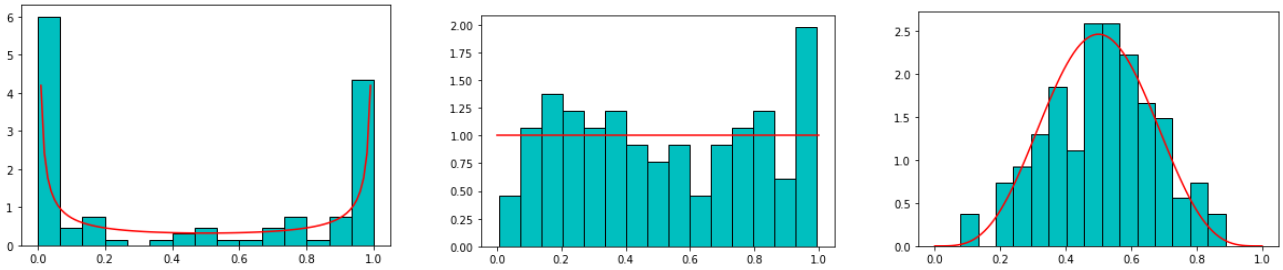
$$f(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}$$

where $B(\alpha, \beta) = \dfrac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ and Γ is the Gamma function.

We consider three different cases: $\alpha = \beta = 0.2$, $\alpha = \beta = 1$, and $\alpha = \beta = 5$. All of these distributions are symmetric around 0.5 as $\alpha = \beta$. For $\alpha = \beta = 0.2$, the probability density function is largest at the endpoints (0 and 1) and decreases towards the middle (0.5). For $\alpha = \beta = 1$, the Beta distribution is identical to the uniform distribution, so when examining the rank of the winners, we would expect a result similar to that of the standard probabilities (10x 0.05, 0.15, ..., 0.95). For $\alpha = \beta = 5$, the probability density function is largest in the middle and smallest at the edges of the interval [0, 1].

We sampled 100 values from each of these distributions. Figure 5.3 shows histograms of these samples. Since we are only sampling 100 values, we want to assess whether we obtained a reasonably accurate sample. To do

this, we also plotted the probability density function in red for each distribution. While the samples are not perfect, they are fairly representative. However, in a real-world scenario, we cannot expect the samples to be perfectly representative either.



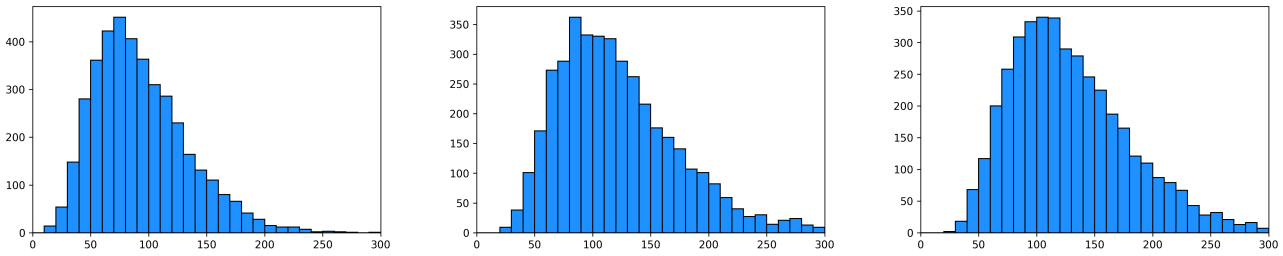
$\alpha = 0.2, \beta = 0.2$ $\alpha = 1, \beta = 1$ $\alpha = 5, \beta = 5$

Figure 5.3: Samples from beta distribution used in Figure 5.4 together with their density (red)

Figure 5.4 shows the ranks of the tournament winners. We can see that the case where $\alpha = \beta = 0.2$ closely resembles Figure 5.2, which makes sense, as most of the sampled probabilities are concentrated towards the left and right ends of the interval [0, 1], with only a few in the middle. In both cases the rank of the contestant most likely to win the tournament is around 75.

When examining the rank of the tournament winner for $\alpha = \beta = 1$, we observe a histogram similar to that in Figure 2.1, which aligns with our expectations.The contestants that win the most often are of rank around 100.

The scenario where $\alpha = \beta = 5$ shows many similarities to scenario 2 from Figure 5.1. Winners are likely to be the 115th most accurate contestant. In both cases, the probabilities are more concentrated in the middle of the interval [0, 1], so it is not surprising that we get similar results.



$\alpha = \beta = 0.2$ $\alpha = \beta = 1$ $\alpha = \beta = 5$

Figure 5.4: Rank tournament winner, probabilities from beta distributions for $0 < \sigma < 0.3$

Thus, taking the success probabilities from a Beta distribution instead of choosing simple values ourselves does not significantly influence the distribution of winners. The main factor influencing the paradox is how much the success probabilities are concentrated in the middle of the interval [0,1], with a higher concentration resulting in an increased effect of the prediction paradox.

5.3 Conclusions

In Section 5.1, we observed that selecting success probabilities in a tournament of 100 events that were skewed toward the left and right ends of the interval [0, 1] caused the rank of the contestant most likely to win to decrease from 100 to around 75. The overall distribution maintained a similar shape, but the mode was shifted more toward the left. Conversely, when the success probabilities were concentrated more towards the middle of the interval [0, 1], we saw the opposite effect: the rank of the contestant most likely to win the tournament increased to 115, and the mode of the distribution shifted more to the right.

Section 5.2 demonstrated that sampling the success probabilities from beta distributions (with parameters $\alpha = \beta = 0.2, 1, 5$) produced an effect similar to equi-spaced success probabilities.

6 Distribution of the predicted probabilities of contestants

6.1 Uniform distribution

In Section 2, we examined a model in which contestants predicted $p \pm \sigma$, each with equal probability, where p is the true probability of an event occurring. David J. Aldous also considered an alternative model in his article (Aldous, 2019), where contestants predict a random value uniformly distributed in $[p - \sigma\sqrt{3},\ p + \sigma\sqrt{3}]$. The factor of $\sqrt{3}$ is introduced to ensure that the variance of the error distribution matches that of the $\pm\sigma$ model. In the case where contestants predict $p \pm \sigma$ with equal probability, the variance is σ^2. By including the $\sqrt{3}$ factor in the uniform model, the variance is likewise σ^2.

If the randomly selected value in $[p-\sigma\sqrt{3},\ p+\sigma\sqrt{3}]$ is greater than 1, we set the predicted value to 1; if it is less than 0, we set it to 0. An alternative method to prevent probabilities larger than 1 or smaller than 0, would have been to truncate the interval $[p - \sigma\sqrt{3},\ p + \sigma\sqrt{3}]$ to lie entirely within $[0, 1]$. We do not have a theoretical argument on why one approach would be better than the other. We only found that the second approach takes significantly longer to simulate, and the results do not differ meaningfully between the two methods. Results for the truncated model can be found in Figure A.1 in Appendix A. Therefore, we analyze only the results from the first method.

Figure 6.1 shows the ranks of the tournament winner for three ranges of σ: $0.05 < \sigma < 0.35$, $0.1 < \sigma < 0.4$, and $0.15 < \sigma < 0.45$, in the uniform model where predicted probabilities are set to 0 or 1 if they fall outside the interval $[0, 1]$. The results in Figure 6.1 closely resemble those in Figure 2.2, which corresponds to the original predict $p \pm \sigma$ model.

For σ in the range $[0.05,\ 0.35]$, the contestant ranked around 50th is most likely to win the tournament in both models. Similarly, as seen in Figure 2.2, for σ in the ranges $[0.1,\ 0.4]$ and $[0.15,\ 0.45]$, the winner is most likely to be among the top 10 best predictors. Thus, by modeling contestant predictions as a random value uniformly in $[p - \sigma\sqrt{3},\ p + \sigma\sqrt{3}]$ instead of using $p \pm \sigma$ does not significantly affect the distribution of winners.

We would like to point out that these results are not consistent with those presented by Aldous in his 2019 article A Prediction Tournament Paradox. We contacted the author regarding this discrepancy, and our results as shown in Figure 6.1 were confirmed.

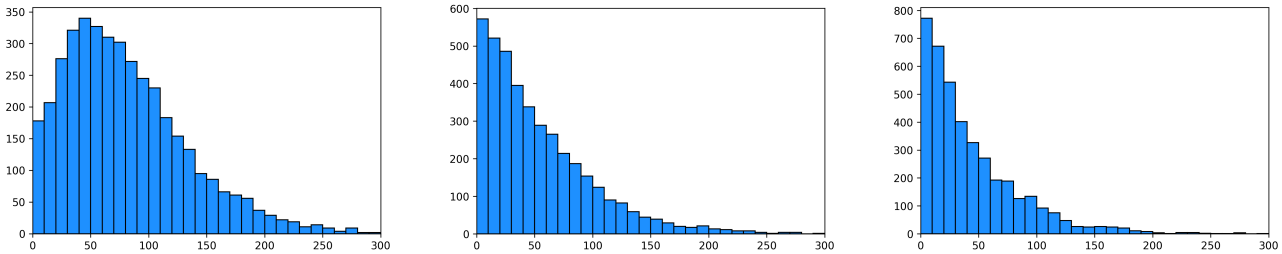


Figure 6.1: Rank tournament winner, uniform error model

6.2 Beta distribution

In Section 5.2, we used the Beta distribution to model success probabilities because it is both versatile and confined to the interval $[0, 1]$. For the same reasons, we now use it to model the predicted probabilities of each contestant. We follow the definition of the Beta distribution given in Definition 5.1.

As we did for the uniform model in Section 6.1, we aim to ensure that the predicted values have a variance of σ^2 and a mean equal to p. When using the Beta distribution, we consider two main approaches to achieve this.

The first method involves choosing fixed values for α and β, which allows us to control the shape of the probability density function. To match the desired mean, we shift the interval $[0, 1]$ accordingly, and to obtain the correct variance, we rescale the interval. This approach is discussed in Section 6.2.1. This can cause contestants to predict values outside of the interval [0, 1]. If this happens, we

The second method does not involve rescaling the interval $[0, 1]$. Instead, we allow α and β to depend on σ in order to match the desired variance, and we shift the interval to ensure the correct mean. This model is

considered in Section 6.2.2.

6.2.1 Fixed α and β

We fix the parameters α and β of the Beta distribution and aim to shift and rescale the interval $[0, 1]$ so that the resulting distribution has a mean of p and a variance of σ^2. The following equation outlines this transformation in detail. Here, X is distributed as $\text{Beta}(\alpha, \beta)$, and Y represents the value predicted by a contestant.

$$Y = \frac{\sigma}{\sqrt{\mathbb{V}\text{ar}(X)}}(X - \mathbb{E}[X]) + p \tag{6.1}$$

To verify:

$$\mathbb{E}[Y] = \mathbb{E}\left[\frac{\sigma}{\sqrt{\mathbb{V}\text{ar}(X)}}(X - \mathbb{E}[X]) + p\right] = \frac{\sigma}{\sqrt{\mathbb{V}\text{ar}(X)}}(\mathbb{E}[X] - \mathbb{E}[X]) + p = p$$

$$\mathbb{V}\text{ar}(Y) = \mathbb{V}\text{ar}\left(\frac{\sigma}{\sqrt{\mathbb{V}\text{ar}(X)}}(X - \mathbb{E}[X]) + p\right) = \mathbb{V}\text{ar}\left(\frac{\sigma}{\sqrt{\mathbb{V}\text{ar}(X)}}X\right) = \frac{\sigma^2}{\mathbb{V}\text{ar}(X)}\mathbb{V}\text{ar}(X) = \sigma^2$$

This is as desired.

The probability density function of the Beta distribution can take various shapes. In this section, we consider four specific cases: two symmetric and two asymmetric. The Beta distribution is symmetric when $\alpha = \beta$ and asymmetric when $\alpha \neq \beta$. For the symmetric cases, we examine $\alpha = \beta = 0.2$ and $\alpha = \beta = 5$. For the asymmetric cases, we look at $\alpha = 2, \beta = 5$ and $\alpha = 1, \beta = 5$. The probability densities of these distributions are visualized in Figure 6.2.

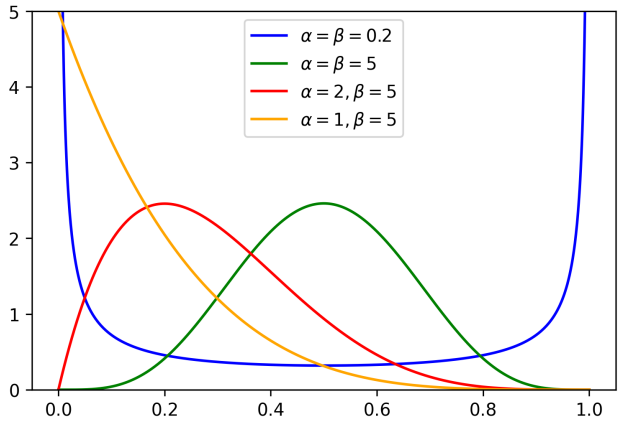
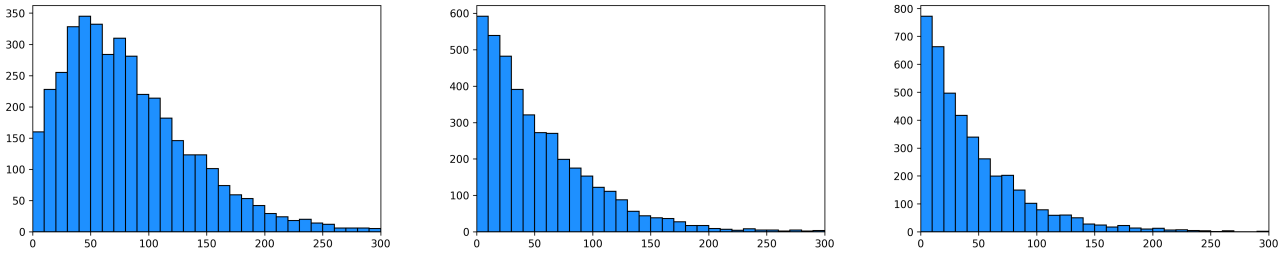


Figure 6.2: Probability density function of the beta distribution for parameters $\alpha = \beta = 0.2$, $\alpha = \beta = 5$, $\alpha = 2, \beta = 5$ and $\alpha = 1, \beta = 5$

In the symmetric case, we chose $\alpha = \beta = 0.2$ because its probability density function places most of its mass at the endpoints of the interval $[0, 1]$. As a result, sampled values are more likely to come from the sides of the interval, though it is still possible to sample a value near the middle. This model represents a continuous variant of the original $p \pm \sigma$ prediction model.
We selected $\alpha = \beta = 5$ for the opposite behavior. In this case, the probability density function has more mass concentrated around the middle of the interval and less near the endpoints.
For the asymmetric case, we chose $\alpha = 2, \beta = 5$ because more, but not all, of its mass is shifted to the left compared to the case where $\alpha = \beta = 5$. A slight bell-like shape is still present, but its peak is no longer at 0.5; instead, it is shifted further to the left. We selected $\alpha = 1, \beta = 5$ to create an even more pronounced shift of mass towards the left. In this case, the bell shape observed for $\alpha = \beta = 5$ disappears entirely, and the probability density function becomes a decreasing function.

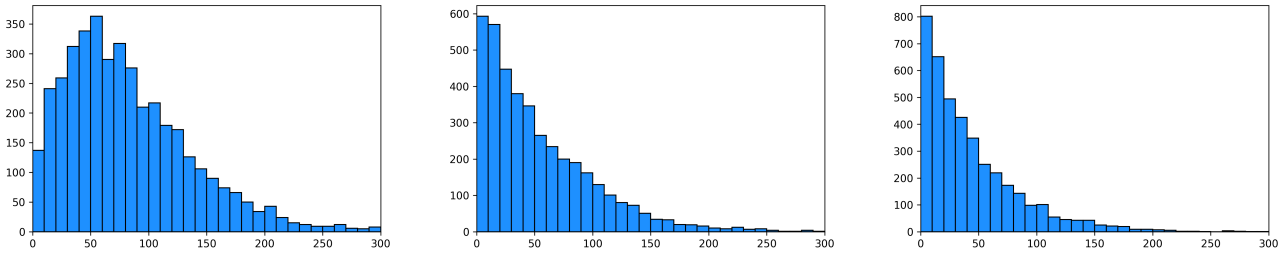
Flipping the values of α and β in these asymmetric cases would mirror the probability density around 0.5. However, we do not consider these mirrored cases because the Brier score is symmetric; overestimating or underestimating the true probability p results in the same penalty. Therefore, examining the mirrored distributions would yield identical results.

Figures 6.3 and 6.4 show the distribution of ranks for the tournament winner for σ in the ranges $[0.05, 0.35]$, $[0.1, 0.4]$, and $[0.15, 0.45]$ in the two symmetric cases, $\alpha = \beta = 0.2$ and $\alpha = \beta = 5$. Figures 6.5 and 6.6 show the same distributions for the asymmetric cases, $\alpha = 2, \beta = 5$ and $\alpha = 1, \beta = 5$. We observe very little difference between Figures 6.3, 6.4, 6.5, and 6.6. These distributions also look very similar to Figure 2.2, which shows the ranks of the winner for the $p \pm \sigma$ model. We can conclude that predicting probabilities from different Beta distributions, whether symmetric or asymmetric, does not cause a significant change in the distribution of winners.



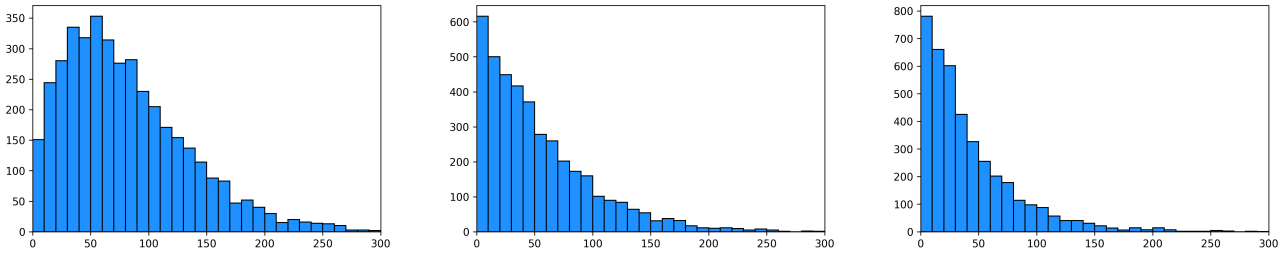
$0.05 < \sigma < 0.35$ $0.1 < \sigma < 0.4$ $0.15 < \sigma < 0.45$

Figure 6.3: Rank tournament winner, symmetric beta error model, $\alpha = \beta = 0.2$



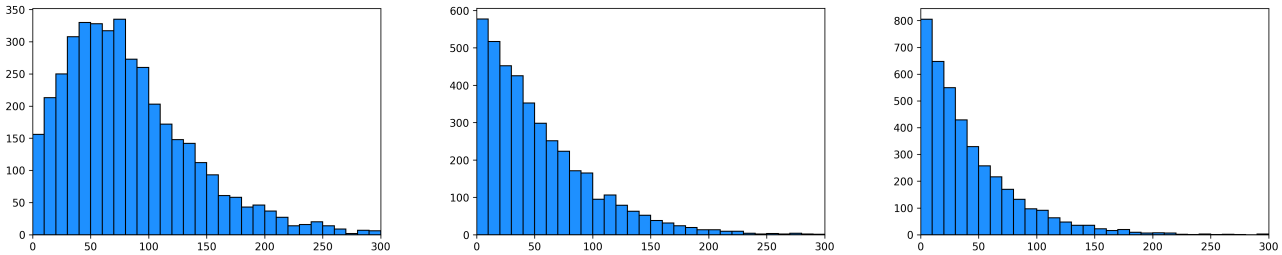
$0.05 < \sigma < 0.35$ $0.1 < \sigma < 0.4$ $0.15 < \sigma < 0.45$

Figure 6.4: Rank tournament winner, symmetric beta error model, $\alpha = \beta = 5$



$0.05 < \sigma < 0.35$ $0.1 < \sigma < 0.4$ $0.15 < \sigma < 0.45$

Figure 6.5: Rank tournament winner, asymmetric beta error model, $\alpha = 2, \beta = 5$



$0.05 < \sigma < 0.35$ $0.1 < \sigma < 0.4$ $0.15 < \sigma < 0.45$

Figure 6.6: Rank tournament winner, asymmetric beta error model, $\alpha = 1, \beta = 5$

6.2.2 α and β dependent on σ

Now, we consider the case where we do not rescale the interval $[0, 1]$. Instead, we let α and β depend on σ to achieve the correct variance and shift the interval to obtain the correct mean. We only focus on the case where

the Beta distribution is symmetric, so $\alpha = \beta$. The variance of the Beta distribution is given by:

$$\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

As $\alpha = \beta$ and we want the variance the be equal to σ^2, we get the following equation

$$\begin{aligned}
\frac{\alpha^2}{(2\alpha)^2(2\alpha+1)} &= \sigma^2 \\
\frac{1}{4(2\alpha+1)} &= \sigma^2 \\
\frac{1}{4\sigma^2} &= 2\alpha+1 \\
\frac{1}{8\sigma^2} - \frac{1}{2} &= \alpha
\end{aligned}$$

To ensure that the variance is equal to σ^2, we need to set $\alpha = \beta = \frac{1}{8\sigma^2} - \frac{1}{2}$. It is important to not that α and β should be larger than 0. This means that to obtain a valid distribution, we require that $\sigma^2 < \frac{1}{4}$ and thus $\sigma < \frac{1}{2}$. We then shift the interval by $-\frac{1}{2} + p$ to achieve a mean of p. Combining these steps gives Equation 6.2, where X is distributed as Beta $\left(\frac{1}{8\sigma^2} - \frac{1}{2}, \frac{1}{8\sigma^2} - \frac{1}{2}\right)$ and Y represents the value predicted by a contestant.

$$Y = X - \frac{1}{2} + p \tag{6.2}$$

Figure 6.7 shows the density of X for $\sigma = 0.1$ (blue), $\sigma = 0.2$ (green), $\sigma = 0.3$ (red), and $\sigma = 0.4$ (yellow). As σ increases, more mass of the probability density function shifts toward the endpoints of the interval $[0, 1]$. This shift is necessary to increase the variance (σ^2). For $\sigma = 0.1$, most of the mass is concentrated in the middle of the interval, and the density function has a bell-shaped curve. As σ increases to 0.4, the situation reverses: the density function is largest near 0 and 1, with the smallest values near 0.5.

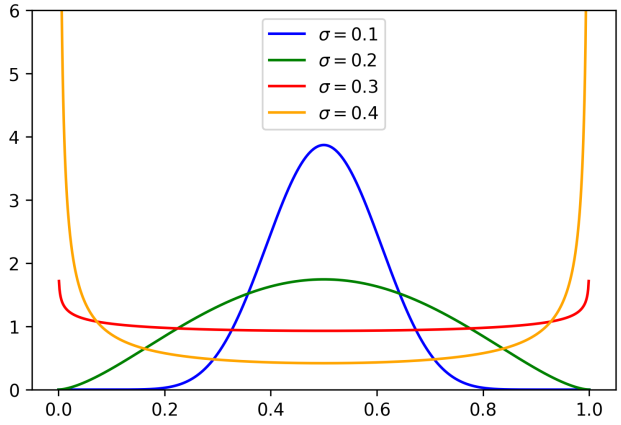
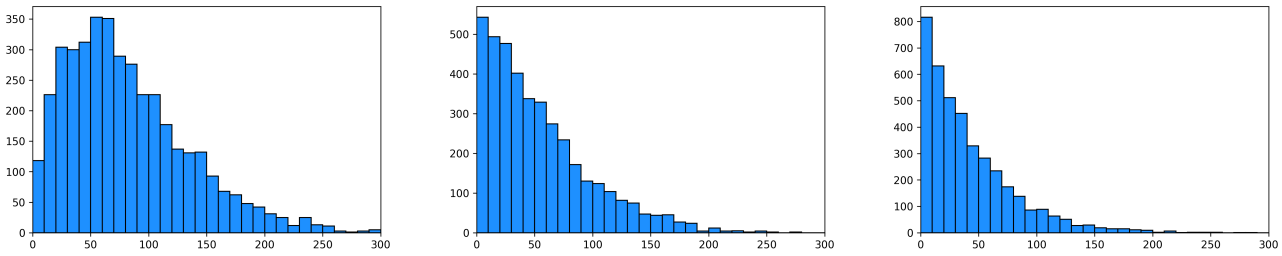


Figure 6.7: Probability density function of the beta distribution for parameters α and β dependent on σ for $\sigma = 0.1, 0.2, 0.3$ and 0.4

Figure 6.8 shows the distribution of winners for this model, with σ in the ranges $0.05 < \sigma < 0.35$, $0.1 < \sigma < 0.4$, and $0.15 < \sigma < 0.45$. As before, we observe that the distribution of winners does not differ significantly from the distribution of winners in the $p \pm \sigma$ model, as presented in Figure 2.2.



$0.05 < \sigma < 0.35$ $\quad$ $0.1 < \sigma < 0.4$ $\quad$ $0.15 < \sigma < 0.45$

Figure 6.8: Rank tournament winner, beta error model, α and β dependent on σ

6.3 Extreme predictions

In Sections 2, 6.1 and 6.2 we looked at a variety of distributions for the values predicted by contestants. However, in all off these cases, contestant do not have a chance of scoring a perfect score of 0. We want to

set up a model where contestants do have a chance of scoring perfectly. We do this by letting contestant make extreme predictions: they always predict 1 or 0. How do we determine which one to predict? Let $q = p \pm \sigma$, like in the original $\pm$ model. A contestant predicts 0 if q is closer to 0, and 1 if q is closer to 1. If $q = 0.5$, we predict 1.

However, if we keep the original distribution of success probabilities of [0.05, 0.15, ..., 0.95], we run into some trouble. Contestants with σ between 0 and 0.05 do not have any randomness in their predictions and make exactly the same predictions. The two probabilities closest to 0.5 are 0.45 and 0.55. So we need σ to be equal to 0.5 or larger to have a chance of predicting 1 if $p = 0.45$ and 0 if $p = 0.55$. The next two probabilities closest to 0.5 are 0.35 and 0.55. For these we need $\sigma \geq 0.15$ to have a chance of predicting the opposite of what p would round to. This pattern continues for $p = 0.25, 0.75$, $p = 0.15, 0.85$ and $p = 0.05, 0.95$. This causes contestants with σ in ranges [0, 0.05], [0.05, 0.15], ..., [0.35, 0.45], to behave exactly the same as other contestant within their range.

To solve this issue we spread the success probabilities equally over the interval [0, 1]. So the success probabilities are as followed: [0, 0.01, 0.02, ..., 0.99]. So we still have 100 events for each tournament. The problem has not been entirely resolved, as contestant with σ in ranges [0, 0.01], [0.01, 0.02],... still behave the same in terms of predictions. However, with the current way the model is set up, there are only 10 contestants in each range instead of 100.

Figure 6.9 shows the distribution of the winner's rank for σ in the intervals $[0, 0.3]$, $[0.05, 0.35]$, $[0.1, 0.4]$, and $[0.15, 0.45]$. This method of extreme predictions yields different results compared to the original $p \pm \sigma$ approach presented in Figures 2.1 and 2.2. The distribution of winners is more dispersed across different ranks, and contestants with higher ranks are more likely to win the tournament in the extreme setting. For example, when considering σ in the range $[0, 0.3]$, Figure 2.1 shows that contestants around rank 100 are most likely to win in the original model, whereas Figure 6.9 indicates that the most likely winner is around the 130th rank.

It is expected that this extreme setting produces different results compared to the original and the uniform and beta settings described in Sections 6.1 and 6.2. In those cases, we fixed the variance of contestants' predictions, whereas in the extreme setting, we cannot control this variance.

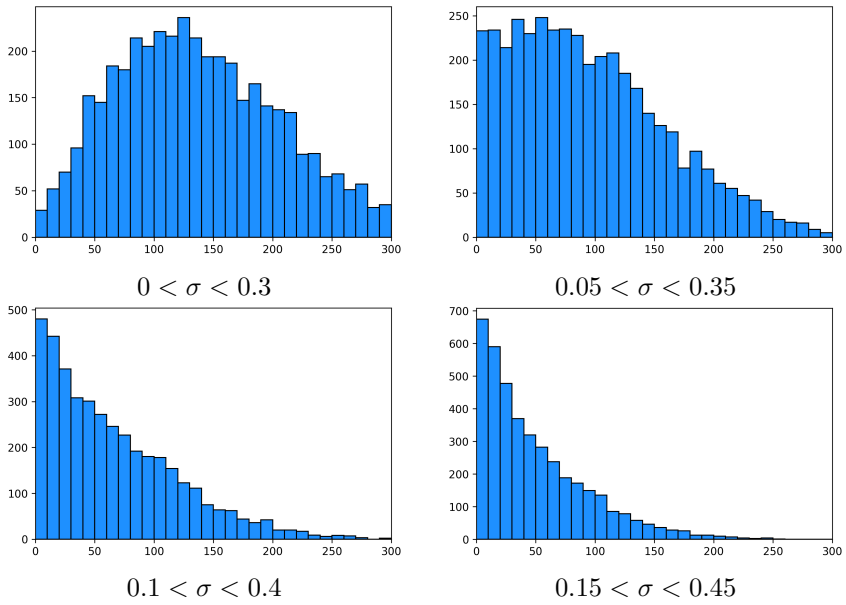


Figure 6.9: Rank tournament winner, Extreme prediction model, success probabilities 0, 0.01, 0.02, ..., 0.99, σ in ranges [0, 0.3], [0.05, 0.35]. [0.1, 0,4], [0.15, 0.45]

This change in variance is also reflected in Figure 6.10. Figure 6.10 shows the average score (blue line), the area between the 0.1 and 0.9 quantiles (dark blue), and the range of scores (light blue) for each contestant with σ in the range $[0, 0.3]$. Comparing Figure 6.10 to Figure 4.7 (the original model), we observe a clear increase in the range of scores for larger values of σ in Figure 4.7, whereas this increase is not clearly present in Figure 6.10. Although scores tend to increase with contestant rank, the difference in score range remains relatively small. Contestants scores differ more greatly in the original model compared to this extreme model. The scores

in the extreme model are relatively similar for each σ. This would explain the more even distribution of winners observed in Figure 6.9.

In the original model, the lowest scores remain roughly constant up to rank 100, after which they begin to increase. In contrast, for the extreme model, the lowest scores remain constant over a broader range of ranks. Additionally, both the highest achieved scores and the average scores are higher in the extreme model compared to the original $p \pm \sigma$ prediction model.

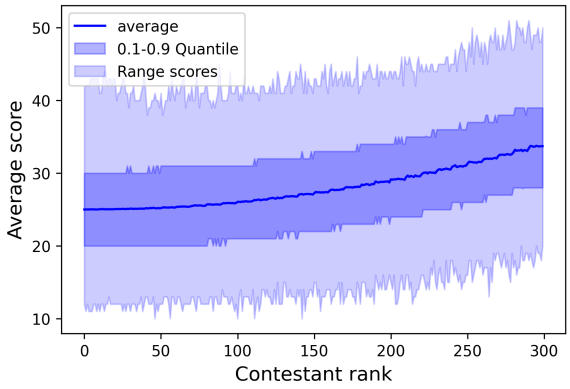


Figure 6.10: Average score, Extreme prediction model, success probabilities 0, 0.01, 0.02, ..., 0.99, σ in range [0, 0.3]

We observed that switching to a model in which all contestants make extreme predictions benefits lower-ranked contestants. However, we are interested in how this extreme prediction strategy compares to the standard $p \pm \sigma$ approach. Specifically, does the extreme prediction strategy outperform the $p \pm \sigma$ strategy? To test this, we set up a tournament in which half of the contestants use the extreme prediction strategy and the other half use the $p \pm \sigma$ strategy. Figure 6.11 shows the ranks of the tournament winners, along with whether they used the $p \pm \sigma$ strategy (blue) or made extreme predictions (orange), for σ values between 0 and 0.3. The results show that the extreme prediction strategy wins significantly less often than the original $p \pm \sigma$ strategy. Therefore, in a mixed tournament setting, the extreme prediction strategy is not recommended.

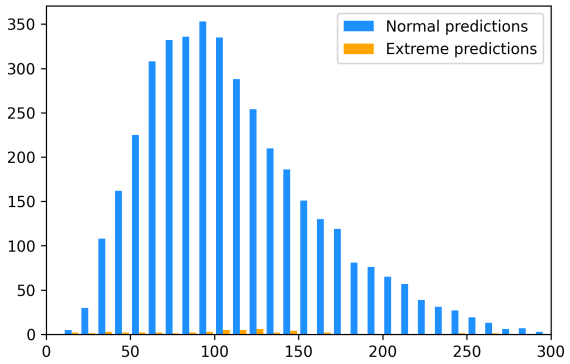


Figure 6.11: Rank tournament winner, mixed tournament with extreme and $p \pm \sigma$ predictions, success probabilities 0, 0.01, 0.02, ..., 0.99, σ in range [0, 0.3]

6.4 Conclusions

In this section, we examined the effects of predicting values from a continuous distribution, rather than using the original $p \pm \sigma$ approach. In Section 6.1, we analyzed contestants predicting a value uniformly from the interval $[p - \sigma\sqrt{3}, p + \sigma\sqrt{3}]$. In Section 6.2, we performed a similar analysis using various parameters of the Beta distribution (both symmetric and asymmetric), and found that when the variance is fixed at σ^2, the distribution of winners does not differ significantly from predicting $p \pm \sigma$. In Section 6.3, we explored a scenario

where contestants make extreme predictions (always predicting 0 or 1). We found that this leads to a more dispersed distribution of winners across the ranks, and contestants with higher ranks have an increased chance of winning. However, in a mixed tournament setting, we observed that this extreme prediction strategy is not advantageous. Contestants using the $p \pm \sigma$ approach are significantly more likely to win.

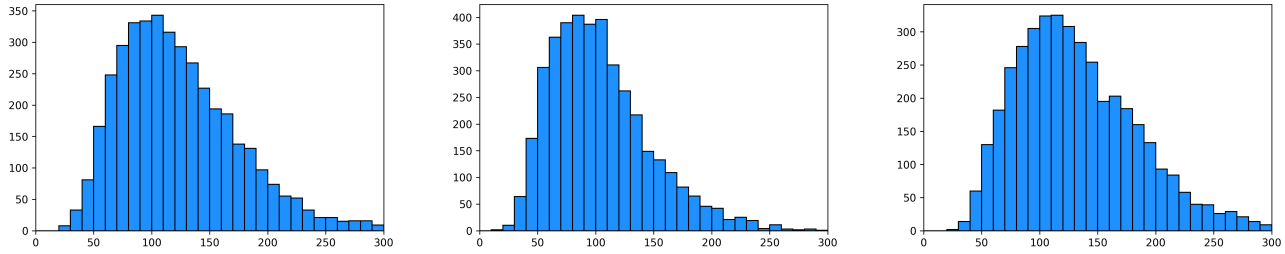
7 Other proper scoring rules

It could also be the case that this effect is specific to the Brier score, and that the Brier score may simply be an unsuitable choice for this type of tournament. To investigate this possibility, we apply different scoring rules to see whether the effect persists. In this section, we compare the Brier, Logarithmic, and Spherical scores, as described in Section 3, all of which are proper scoring rules. The Brier score is symmetric, both the Spherical and Logarithmic scoring rules are not.

After this, we will consider the Pseudospherical score and the Power score, both which have a parameter β and both are nonsymmetric scoring rules. We will compare results for different values of β.

7.1 Brier, Logarithmic and Spherical Score

Figure 7.1 shows the results of simulations using the Brier, Logarithmic and Spherical Score for σ between 0 and 0.3. We see that for both the Logarithmic and Spherical scores, the top 10 most accurate forecasters also fail to consistently win the tournament. However, the Logarithmic score appears to perform slightly better: contestants around rank 80 win the tournament most frequently, compared to around rank 100 for the Brier score and around rank 110 for the Spherical score. Additionally, for both the Brier and Spherical scores, contestants ranked between 240 and 300 occasionally win the tournament. This almost never occurs with the Logarithmic score.



Brier Score | Logarithmic Score | Spherical Score

Figure 7.1: Rank tournament winner Brier, Logarithmic and Spherical score for $0 < \sigma < 0.3$

In addition to examining the rank of the winner, the average finish position per contestant provides further insight. Figure 7.2 presents these results. As seen previously in Figure 7.1, the most accurate predictor performs worse under the Brier and Spherical scores. For the Logarithmic score, the average finish position of the most accurate forecaster is around 30, whereas for the Brier and Spherical scores, it is approximately 35 and 40 respectively. The lowest average finish position overall is achieved by the 34th most accurate contestant under the Brier score, the 29th under the Logarithmic score, and the 38th under the Spherical score. This suggests that the Logarithmic score performs best in terms of rewarding forecasting skill, at least at the top end.

However, a closer look at the average finish positions for contestants ranked 50 to 300 reveals some unusual behavior for the Logarithmic score. While the Brier and Spherical scores exhibit a roughly smooth increase in average rank from contestant 50 to 300, the Logarithmic score does not follow this pattern. There are big jumps in average finish position at ranks 50, 150, and 250. Between ranks 150 and 250, as well as between 250 and 300, there is little to no visible increase in average finish position. This will make it harder to distinguish between these contestants within these rank intervals.

Figure 7.3 shows the average contestant rank per finish position for each scoring rule. The lowest average rank overall is achieved at the 35th finish position under the Brier score, the 31st under the Logarithmic score, and the 40th under the Spherical score. The average rank of the tournament winner is 122 for the Brier score, 103 for the Logarithmic score, and 132 for the Spherical score. Once again, the Logarithmic score appears to perform the best, at least for the lower finish positions. However, when we examine higher finish positions (50 and above), the Logarithmic score exhibits some irregular behavior. While both the Brier and Spherical scores increase roughly linearly beyond this point, the Logarithmic score increases more steeply up to position 150, and then begins to decrease, resulting in a noticeably erratic or "wobbly" pattern.

This irregularity may be explained by the fact that both the Brier and Spherical scores are bounded, while the Logarithmic score is unbounded. Under the Logarithmic scoring rule, contestants can receive a score of

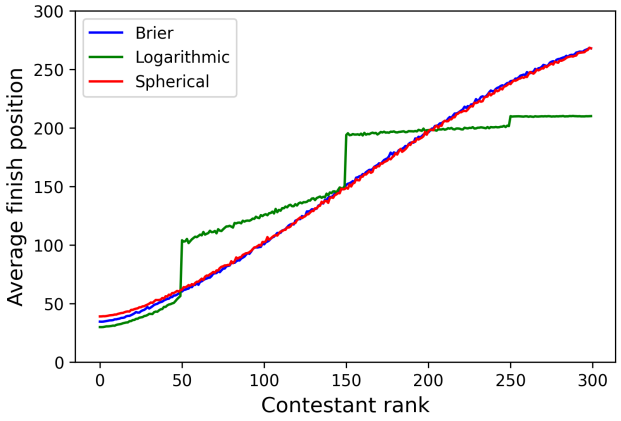


Figure 7.2: Average finish position per contestant for the Brier, Logarithmic and Spherical score, 300 contestants, $0 < \sigma < 0.3$

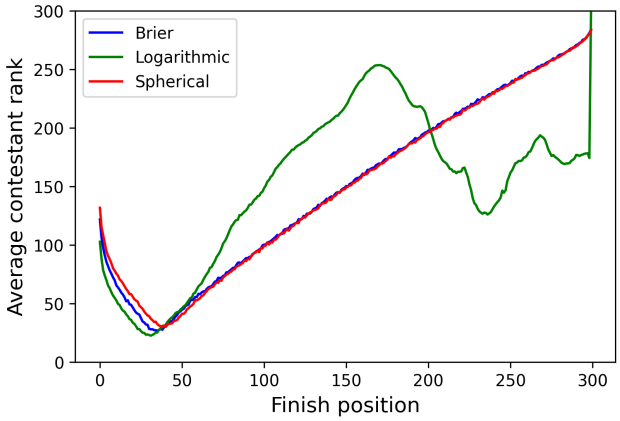


Figure 7.3: Average rank for each finish position for the Brier, Logarithmic and Spherical score, 300 contestants, $0 < \sigma < 0.3$

infinity. This is particularly relevant for lower ranked contestants, those with larger σ, who are more likely to predict extreme probabilities (0 or 1) for events with true probabilities near 0 or 1.

If such a contestant predicts 0 and the event occurs (or predicts 1 and the event does not occur), the logarithmic penalty becomes infinite for that event. Since total score is the sum of individual event scores, even a single infinite score results in a total score of infinity. Consequently, that contestant is automatically assigned (shared) last place. This makes the Logarithmic score less accurate for distinguishing between lower ranked contestants.

The jumps observed in Figure 7.2 can also be explained with this reasoning. The true success probabilities p of the events are in the set $\{0.05, 0.15, 0.25, \ldots, 0.95\}$, and contestants predict values of the form $p \pm \sigma$. For $\sigma = 0.05, 0.15, 0.25$, which correspond to ranks 50, 150, and 250 respectively, there is an increase in the number of contestants who predict values 0 or 1.

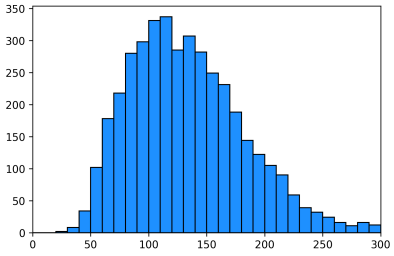
This leads to an increase in scores that evaluate to ∞. As a result, it becomes easier to distinguish between contestants at adjacent ranks around these points (between ranks 49 and 50, 149 and 150, and 249 and 250), causing noticeable jumps in average finish position.

The jumps in Figure 7.2 can also be explained with this reasoning. As the true success probabilities p of the events are in {0.05, 0.15, 0.25, ..., 0.95} and contestants predict $p \pm \sigma$. For $\sigma = 0.05, 0.15, 0.25$, which correspond to rank 50, 150 and 250 respectively, we get an increase in contestants who are able to predict 0 or 1. So we also get an increase in scores that take value ∞. This makes it easier to distinguish between a contestant of rank 49 and 50, 149 and 150, and 249 and 250. This causes a jump in average finish position.

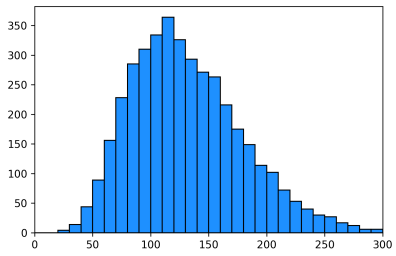
Since true probabilities close to 0 or 1 can distort the results, we now consider a scenario in which the true probabilities are concentrated near the center of the interval [0,1]. Specifically, we use the middle scenario described in Section 5.1, where the lowest true probability is 0.40 and the highest is 0.60. As σ lays between 0 and 0.3, none of the contestants should predict 0 or 1 as the true probability. As a result, none of the contestants should be getting a score of infinity.

To compare the Brier, Logarithmic and Spherical score, we simulated this scenario for all three scoring rules. The results of this are presented in Figure 7.4. We observe that the Logarithmic score now performs comparably to the Brier and Spherical score. For all three scoring rules, contestants around rank 115 are most likely to win the tournament. Additionally, the overall distribution of winners appears to be roughly similar across all three scores.

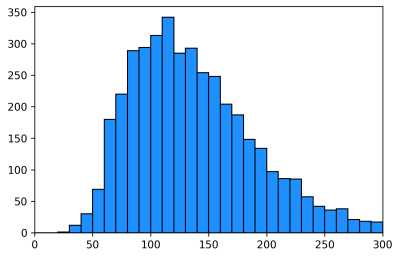
Figure 7.5 shows the average contestant rank per finish position under the Logarithmic scoring rule for both the standard true probability distribution (green) and the middle scenario (yellow). We can see that, in the middle scenario, the Logarithmic score behaves as expected for contestants ranked above 50: higher finish positions have higher average rank. As discussed above, this behavior is not seen in the case of the standard true probability distribution (green line). Figure 7.6 shows the average rank per finish position for each scoring rule in the middle scenario. In contrast to Figure 7.3, we now see that the average finish position per contestant is nearly identical across the Brier, Logarithmic, and Spherical scores. The advantage the Logarithmic score



Brier Score



Logarithmic Score



Spherical Score

Figure 7.4: Rank tournament winner for the Brier, Logarithmic and Spherical score for $0 < \sigma < 0.3$ for success probabilites in the middle interval [0,1]

provided for more accurate contestants is gone.

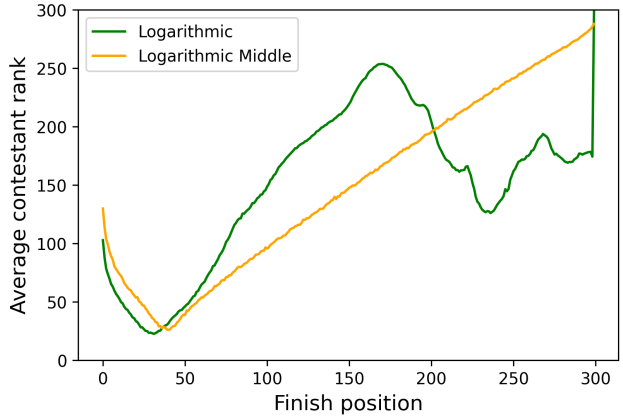


Figure 7.5: Average contestant rank per finish position for the Logarithmic score with standard true probabilities and middle probabilities

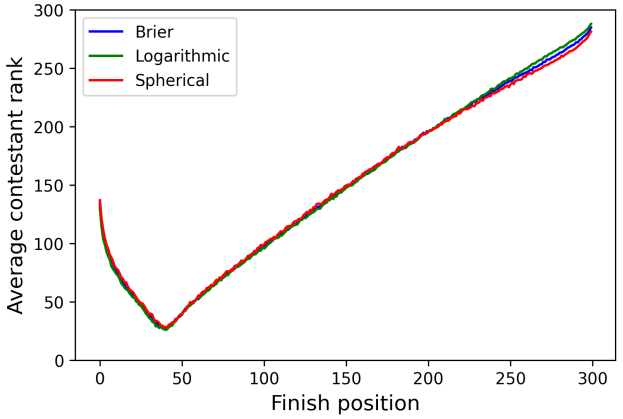


Figure 7.6: Average contestant rank per finish position for the Brier, Logarithmic and Spherical score with true probabilities as in the middle scenario

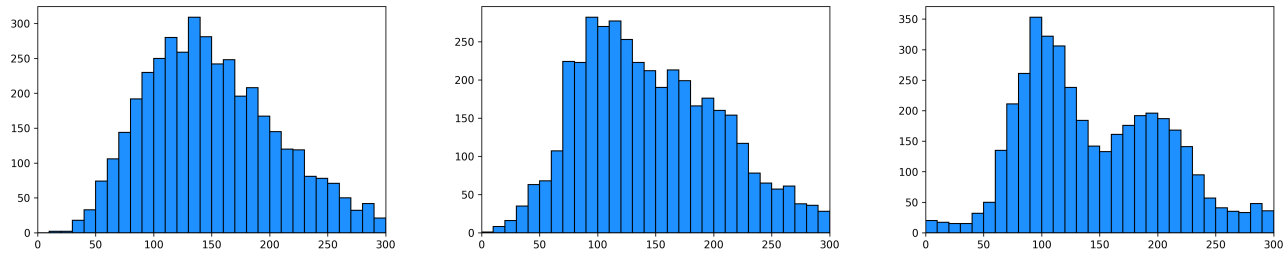
7.2 Pseudospherical Score

A variation of the Spherical score is the Pseudospherical score with parameter β. For $\beta = 2$, the Pseudospherical score is equal to the Spherical score. Like the Spherical score, the Pseudospherical score is not symmetric. In Section 7.1, we saw that the Spherical score did not perform significantly better than the Brier or Logarithmic scores, but the performance of the Pseudospherical score may improve for values of β other than 2.

Figure 7.7 shows the ranks of the tournament winner for $\beta = 5$, 10, and 20. We again kept σ between 0 and 0.3. For all these values of β, we see that the more accurate contestants (ranks 0–40) win more often compared to the Spherical score in Figure 7.1. However, the less accurate contestants (ranks 250–300) also win more often compared to Figure 7.1.

For $\beta = 5$, contestants around rank 135 are most likely to win the tournament, which is worse than for the Spherical score. For $\beta = 10$, this improves slightly—contestants around rank 110 are most likely to win. However, this peak is much lower, and the winners appear to be more evenly spread between ranks 70 and 220. The number of winners only slightly decreases as contestant rank approaches 220. The case in which $\beta = 20$ has a similar issue. Contestants around rank 100 are most likely to win the tournament, which is better than what we observed for the Spherical score. However, for $\beta = 20$, we also observe a second peak around rank 200. This peak is slightly more than half as high as the first, meaning the tournament winner is still quite likely to not be a very accurate forecaster.

Figure 7.8 shows the average score of each contestant for the Spherical score ($\beta = 2$, in blue) and the Pseudospherical score for $\beta = 5$ (green), $\beta = 10$ (red), and $\beta = 20$ (yellow). We can see that the shape of the



$\beta = 5$ $\beta = 10$ $\beta = 20$

Figure 7.7: Rank tournament winner Pseudospherical score with $\beta = 5, 10, 20$ for $0 < \sigma < 0.3$

average score for $\beta = 2$, 5, and 10 is similar. For $\beta = 2$, the increase is slightly steeper, and the score values are lower than those for $\beta = 5$ and 10. The curve for $\beta = 20$ generally follows a similar trend to $\beta = 10$), but displays some waviness. In particular, there is a relatively large, mostly flat region for contestants ranked between 170 and 230. This could explain the second peak of winners observed in Figure 7.7, as it would be harder to distinguish between contestants ranked within this range by means of score, but easier to distinguish between contestants ranked between 170 and 230 and those ranked outside this interval.

We also examine the average finish position per contestant. Figure 7.9 shows this for $\beta = 2$ (blue), $\beta = 5$ (green), $\beta = 10$ (red), and $\beta = 20$ (yellow). Similar to the average scores shown in Figure 7.8, the average finish position increases with contestant rank. This increase is steeper for $\beta = 2$ and becomes more gradual as β increases. For $\beta = 20$, we also observe the same waviness that appeared in the average score for $\beta = 20$. Thus, the regular Spherical score outperforms the Pseudospherical score with larger values of β in terms of average finish position per contestant.

In Figure 7.10, the average contestant rank per finish position is shown for the Pseudospherical score with parameters $\beta = 2$ (blue), $\beta = 5$ (green), $\beta = 10$ (red), and $\beta = 20$ (yellow). We observe that the average rank of the winner is similar for $\beta = 5$, 10, and 20, at around 150, whereas for $\beta = 2$, the average rank is lower, around 125.

For the Spherical score, there is a clear dip in average rank around finish position 40. For $\beta = 5$, 10, and 20, this dip is less pronounced, and its lowest point occurs at a higher finish position. Therefore, the Pseudospherical score with $\beta = 5$, 10, and 20 also performs worse than the Spherical score in this regard.

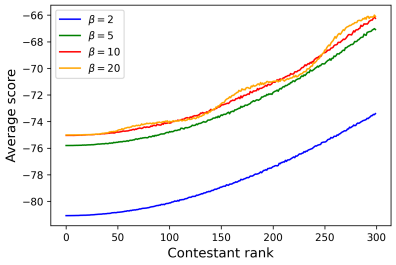


Figure 7.8: Average score per contestant for the Pseudospherical score for $\beta = 2, 5, 10, 20$

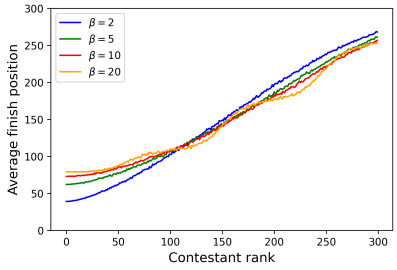


Figure 7.9: Average finish position per contestant for the Pseudospherical score for $\beta = 2, 5, 10, 20$

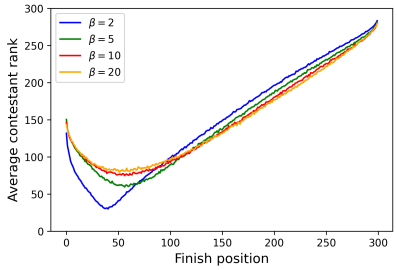
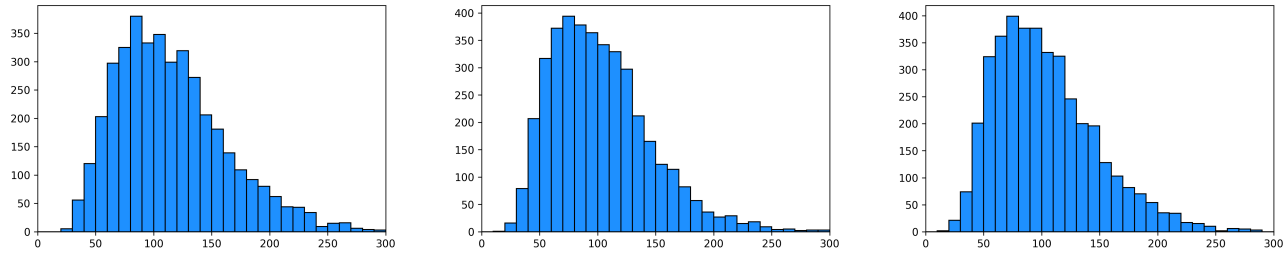


Figure 7.10: Average contestant rank per finish position per contestant for the Pseudospherical score for $\beta = 2, 5, 10, 20$

For values of β larger than 2, the Pseudospherical scoring rule actually performs worse than the Spherical score in terms of accurate winners, average finish position per contestant rank, and average rank per finish position. However, the Pseudospherical score is a proper scoring rule for $\beta > 1$. Therefore, we can also examine the effect of choosing smaller values of β (while remaining above 1) to analyze whether this improves performance.

Figure 7.11 shows the ranks of the tournament winner for $\beta = 1.5$, $\beta = 1.1$, and $\beta = 1.01$. In the cases of $\beta = 1.5$, the tournament winner is most likely to come from around the 100th-ranked contestant. This result is nearly identical to that of the Brier score in Figure 7.1.

In the case where $\beta = 1.1$ and $\beta = 1.01$, we observe a slight improvement. All ranks are shifted slightly to the left compared to the regular Spherical score or the Brier sccore, and the tournament winner is most likely to be around the 90th most accurate contestant. Thus, we gain some performance by decreasing the value of β.



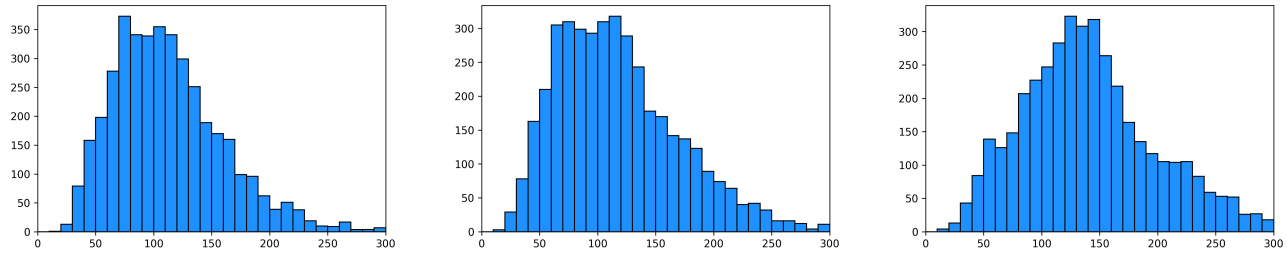
$\beta = 1.5$ $\beta = 1.1$ $\beta = 1.01$

Figure 7.11: Rank tournament winner Pseudospherical score with $\beta = 1.5, 1.1, 1.01$ for $0 < \sigma < 0.3$

7.3 Power Score

The Pseudospherical score lead to some improvements, but maybe we can do better, so we now turn to the Power score. For $\beta = 2$, the Power scoring rule is equivalent to an affine transformation of the Brier score. This means that using the Power score with $\beta = 2$ results in the same tournament winners as the Brier score. To assess whether increasing β affects performance, we examine higher values of β.

Figure 7.12 displays the ranks of the tournament winner for $\beta = 5$, 10, and 20. For $\beta = 5$ and $\beta = 10$, the most likely tournament winner remains around rank 100. However, for $\beta = 5$, contestants ranked between 70 and 120 are nearly equally likely to win, and for $\beta = 10$, this range broadens to ranks 60–130. This is a wider range compared to the Brier score, which shows a winner range of 80–120, as seen in Figure 2.1. Figure 7.12 also shows that the rank of the contestant most likely to win the tournament increases from 100 to 130 when $\beta = 20$.



$\beta = 5$ $\beta = 10$ $\beta = 20$

Figure 7.12: Rank tournament winner Power score with $\beta = 5, 10, 20$ for $0 < \sigma < 0.3$

For the Pseudospherical score, we observed multiple peaks in Figure 7.7 for $\beta = 20$. This behavior is not seen for the Power score with parameter $\beta = 20$. It might be the case that such a pattern only emerges for larger values of β. Figure 7.13 shows the distribution of the ranks of the tournament winner for the Power score with parameter $\beta = 50$. In this case, we do observe multiple peaks. The contestant most likely to win the tournament is at around rank 160, which also corresponds to the tallest peak by a significant margin in Figure 7.13. Additionally, there are two smaller peaks: one around rank 60 and another around rank 260.

An explanation for this behavior can be found in Figures 7.14, 7.15, and 7.16. Figure 7.14 shows the average score per contestant, Figure 7.15 the average finish position per contestant, and Figure 7.16 the average rank per finish position for $\beta = 5$ (blue), $\beta = 10$ (green), $\beta = 20$ (red), and $\beta = 50$ (yellow).

In Figure 7.14, we observe that the average score increases with contestant rank for all values of β. However, as β increases, a noticeable waviness begins to emerge. For $\beta = 5$ and $\beta = 10$, this effect is minimal. For $\beta = 20$, it becomes more pronounced, and for $\beta = 50$, it is quite extreme. Specifically, for $\beta = 50$, we observe that around ranks 50, 150, and 250, the average score plateaus, followed by a steep increase roughly 50 ranks

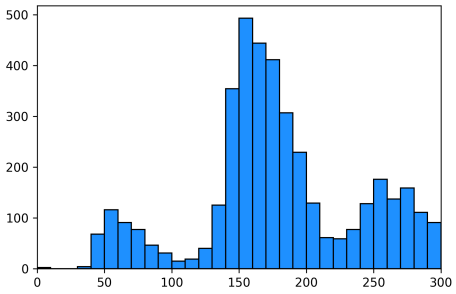


Figure 7.13: Rank tournament winner Power score with $\beta = 50$ for $0 < \sigma < 0.3$

later. This pattern could explain the three peaks observed in Figure 7.13, as each plateau corresponds to one of the peaks. These plateaus make it more difficult to distinguish between contestants within the same plateau by means of score, while the steep increases between plateaus make it easier to differentiate between contestants from different plateaus.

This behavior is also reflected in Figure 7.15. For $\beta = 5$ and $\beta = 10$, the average finish position increases smoothly as contestant rank increases. For $\beta = 20$ and $\beta = 50$, we observe the same plateau followed by a steep increase, similar to the pattern seen in Figure 7.14. However, there is a noticeable difference between $\beta = 20$ and $\beta = 50$. For $\beta = 20$, the average finish position still increases with contestant rank. In contrast, for $\beta = 50$, this is no longer the case. Specifically, for contestants ranked between 0 and 50, the average finish position actually decreases as contestant rank increases.

In Figure 7.16, we can also observe that the average rank of the winner increases as β increases. Specifically, the average rank of the winner is approximately 110, 115, 140, and 170 for $\beta = 5$, 10, 20, and 50, respectively. After this point, the average rank decreases until around finish positions 30–50, depending on the value of β. At this low point, the average rank is lowest for $\beta = 5$, followed by $\beta = 10$, $\beta = 20$, and finally $\beta = 50$. Beyond this range, the average rank increases smoothly with finish position for $\beta = 5$, 10, and 20. For $\beta = 50$, however, we observe a slight wavy pattern, which was also observed in Figures 7.14 and 7.15.

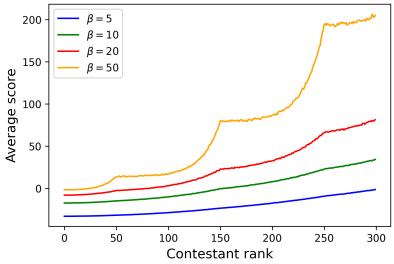


Figure 7.14: Average score per contestant for the Power score for $\beta = 5, 10, 20, 50$

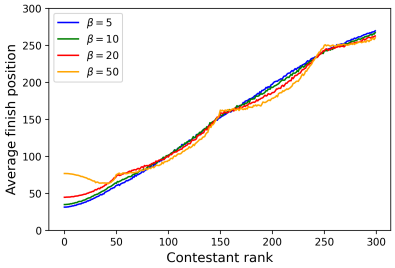


Figure 7.15: Average finish position per contestant for the Power score for $\beta = 5, 10, 20, 50$

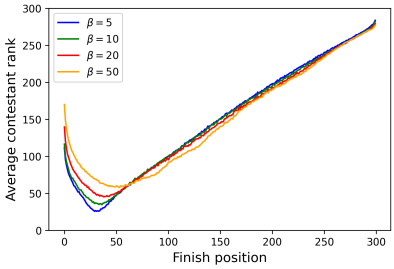
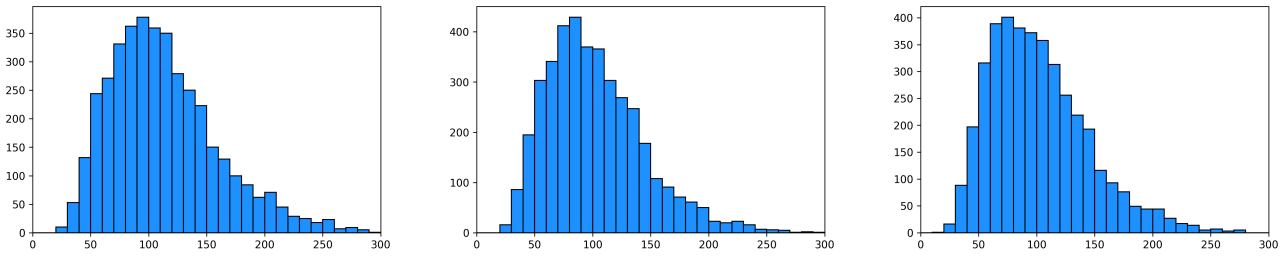


Figure 7.16: Average rank per finish position for the Power score for $\beta = 5, 10, 20, 50$

As observed with the Pseudospherical score, increasing the value of β also leads to a decline in performance for the Power score. This is evident in the distribution of winner ranks, the average finish position per contestant, and the average rank per finish position. The Power score is a proper scoring rule for all $\beta > 1$. This allows us to also examine the effect of choosing values of β closer to 1.

Figure 7.17 shows the distribution of winner ranks for $\beta = 1.5$, 1.1, and 1.01. For $\beta = 1.5$, we observe no significant difference compared to Figure 2.1, where the Brier scoring rule is used. However, for $\beta = 1.1$ and 1.01, there is some improvement. The rank of the contestant most likely to win the tournament decreases from around 100 to around 80. The results in Figure 7.17 for $\beta = 1.1$ and 1.01 are quite similar to those in Figure 7.1 for the Logarithmic score.

We are curious whether the observed decrease continues as β decreases further. Figure 7.18 shows the ranks of the tournament winner for $\beta = 1.00001$. We observe no significant difference between Figure 7.18 and the cases where $\beta = 1.1$ and $\beta = 1.01$ in Figure 7.17.



$\beta = 1.5$ $\beta = 1.1$ $\beta = 1.01$

Figure 7.17: Rank tournament winner Power score with $\beta = 1.5, 1.1, 1.01$ for $0 < \sigma < 0.3$

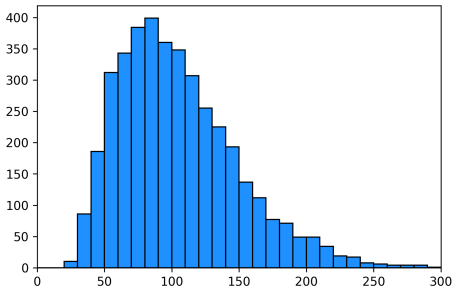


Figure 7.18: Rank tournament winner Power score with $\beta = 1.00001$ for $0 < \sigma < 0.3$

For further analysis, we examine the average score, finish position, and rank. Figure 7.19 shows the average score per contestant, Figure 7.20 shows the average finish position per contestant, and Figure 7.21 shows the average rank per finish position for $\beta = 1.5$ (blue), $\beta = 1.1$ (green), $\beta = 1.01$ (red), and $\beta = 1.00001$ (yellow).

In Figure 7.19, we observe that for $\beta = 1.5$, the average score increases fairly smoothly as contestant rank increases. However, for $\beta = 1.1$, 1.01, and 1.00001, the increase is not as smooth. Instead, there is a stepwise increase with steps at ranks 50, 150, and 250. This stepwise increase is also visible in Figure 7.20, where the average finish position for each contestant follows a similar pattern. For $\beta = 1.5$, the increase remains smooth. For $\beta = 1.1$, 1.01, and 1.00001, the average finish position per contestant is nearly identical. The shape of the curve is also similar to that of the Logarithmic score in Figure 7.2, although it is a bit more stretched out. In Figure 7.2, higher ranking contestants have a lower average finish position compared to Figure 7.20. This difference can be attributed to the fact that the Logarithmic score is unbounded and that many contestants receive a score of infinity. This is further explained in Section 7.1. For the Logarithmic score, many contestants share last place, which is not always position 300 but depends on how many contestants receive the same highest score. For tied values, the average finish position is used. When a large number of contestants tie for last place, this average decreases. The Power score does not have this issue, as it is bounded. Significantly less contestants will share last place.

We observe nearly identical results for $\beta = 1.1$, 1.01, and 1.00001 in Figure 7.21. The average rank for the winner is slightly lower for these values of β compared to $\beta = 1.5$. Between finish positions 0 and 30, the average contestant rank decreases similarly for all values of β. However, after finish position 30, the average rank for $\beta = 1.5$ increases almost linearly, while for $\beta = 1.1$, 1.01, and 1.00001, a certain waviness begins to emerge. Nonetheless, this behavior is much more stable than the erratic fluctuations observed in Figure 7.3 for the Logarithmic score.

In Section 7.1, we saw that the stepwise increase in average finish position at contestant ranks 50, 150, and 250 for the Logarithmic score was caused by truncating the predicted probabilities to the interval [0, 1]. We can avoid this truncation by choosing true success probabilities in the middle of the interval [0, 1], as described in Section 5.1. In this case, the advantage of the Logarithmic score disappeared completely.

As we observed a similar stepwise increase at ranks 50, 150, and 250 in average finish position for the Power score with $\beta = 1.1$, 1.01, and 1.00001, we wanted to investigate whether a similar effect occurs. To do this, we used the same true success probabilities in the middle of the interval [0, 1], as described in Section 5.1. We focus on the case where $\beta = 1.01$, as $\beta = 1.1$ and $\beta = 1.00001$ produced similar results. These results are presented

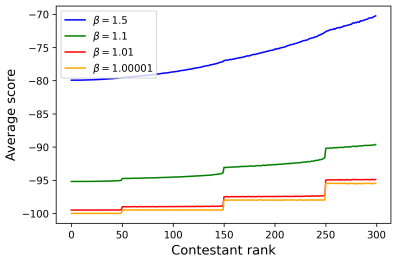


Figure 7.19: Average score per contestant for the Power score for $\beta = 1.5, 1.1, 1.01$

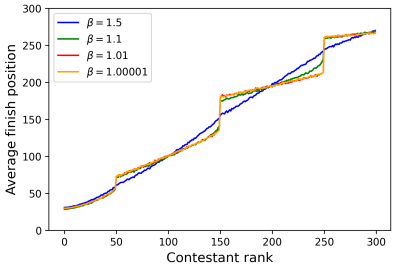


Figure 7.20: Average finish position per contestant for the Power score for $\beta = 1.5, 1.1, 1.01$

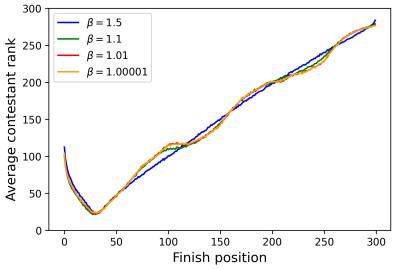


Figure 7.21: Average rank per finish position for the Power score for $\beta = 1.5, 1.1, 1.01$

in Figure B.1 in Appendix B. Figure 7.22 shows the ranks of the winner for this scenario. We observe little difference between Figure 7.22 and the middle scenario in Figure 5.1, which uses the Brier score. This indicates that the benefit gained by the Power score disappears once the success probabilities shift towards the middle of the interval [0, 1].

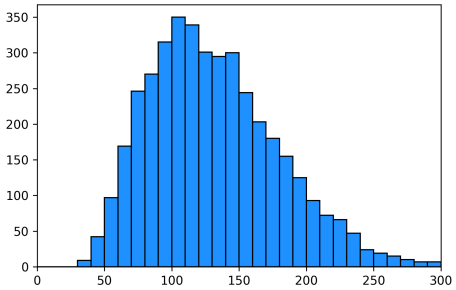


Figure 7.22: Rank tournament winner Power score with $\beta = 1.01$, true success probabilities in the middle of [0, 1], for $0 < \sigma < 0.3$

7.4 Conclusions

In this section, we observed that the choice of scoring rule matters a lot. The Logarithmic scoring rule provided better results when looking at the tournament winner, but performed worse in terms of average finishing position for lower ranked contestants. The benefit of the Logarithmic score also disappears when the true probabilities are concentrated more towards the middle of the interval [0,1].

We also saw that the Pseudospherical score did not perform better than the Spherical score for $\beta > 2$. For β closer to 1, the there was some increase in performance. The Power score also performed worse for larger values of β. For β closer to 1, a significant improvement was observed. However, like for the Logarithmic score, this improvement disappeared when the true probabilities are concentrated more towards the middle of the interval [0,1].

8 Multi-category random variables

In Sections 2, 4, 5, 6, and 7, we focused exclusively on tournaments involving binary events, where each outcome was either a success or a failure. While convenient for simulations, this binary framework imposes a significant limitation on the types of events that can be modeled and predicted. In this section, we extend the analysis to accommodate multi-categorical random variables. This generalization allows for a wider variety of prediction tasks. For instance, instead of forecasting whether it will rain or not, a contestant may be asked to assign probabilities to multiple possible outcomes such as no rain, light rain, or heavy rain.

8.1 Choosing Success Probabilities

Before we can set up such a tournament, we need to choose the true probabilities p_i for $i \in 0, \ldots, C$. Here, p_i represents the probability that the random variable takes the value i. In Section 5.2, we saw that, in the binary case, there was no significant difference between sampling the success probabilities from a uniform distribution and setting them to $10 \times [0.05, 0.15, \ldots, 0.95]$. Therefore, we will use the uniform approach here as well.

If we have $C + 1$ categories, we sample C values from a uniform $[0, 1]$ distribution. We then sort these sampled values, and the success probabilities are defined as the distances between consecutive sorted values. If X_i are the sampled values, this gives us $p_i = X_{(i+1)} - X_{(i)}$. We define $X_{(0)} = 0$ and $X_{(C+1)} = 1$ to ensure that the resulting probabilities sum to 1. A visualization of this process is shown in Figure 8.1.

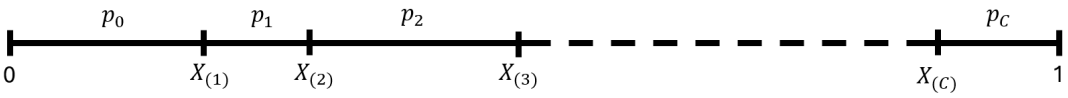


Figure 8.1: Visualization of division of success probabilities

Through repeated simulations, we found that the distribution of winners is more sensitive to the sampled success probabilities in the multi-category case compared to the binary case. To account for this sensitivity, we do not rely on a single sample of success probabilities across all simulated tournaments. Instead, we resample the success probabilities multiple times throughout the simulation process.

8.2 Predicted Probabilities

Next, we discuss what predictions contestants should make. The intuitive way to set up a tournament with $C + 1$ categories is to let contestants predict $q_i = p_i \pm \sigma$ for $i \in 0, \ldots, C - 1$, where $+\sigma$ and $-\sigma$ are chosen independently and with equal probability for each i. We then choose q_C such that $\sum q_i = 1$.

However, in the binary case, a contestant's prediction differed from the true probabilities by σ for two values. In contrast, in a tournament with $C + 1$ categories, the prediction differs by σ for at least C probabilities. The deviation for q_C depends on the predictions made for $q_0, ..., q_{C-1}$. In this sense, the total deviation between the predicted and true distributions increases as the number of categories grows.

To address this issue, we use a formal measure of the distance between two distributions: the total variation distance.

Definition 8.1 (Total variation Distance)**.** Let $(\Omega, \mathcal{F})$ be a measurable space and $\mathbb{P}$ and $\mathbb{Q}$ probability measures on this space. Then the total variation distance between $\mathbb{P}$ and $\mathbb{Q}$ is defined as followed:

$$\delta(\mathbb{P}, \mathbb{Q}) = \sup_{A \in \mathcal{F}} [\mathbb{P}(A) - \mathbb{Q}(A)]$$

This the largest absolute difference that $\mathbb{P}$ and $\mathbb{Q}$ assign to the same event. (Kolmogorov, 1963)

If densities p and q exist for $\mathbb{P}$ and $\mathbb{Q}$, the total variation distance can be computed as follows:

$$\delta(\mathbb{P}, \mathbb{Q}) = \frac{1}{2} \int |p(x) - q(x)| \, \mathrm{d}x \tag{8.1}$$

(Tsybakov, 2009)

In our multi-category case this becomes

$$\delta(\mathbb{P}, \mathbb{Q}) = \frac{1}{2} \sum_{i=0}^{C} |p_i - q_i| \tag{8.2}$$

So, for our original model with binary random variables, the total variation distance is

$$\delta(\mathbb{P}, \mathbb{Q}) = \frac{1}{2}(|1 - p \mp \sigma - (1-p)| + |p \mp \sigma - p|) = \frac{1}{2}(\sigma + \sigma) = \sigma$$

If we follow the intuitive approach where contestants predict $q_i = p_i \pm \sigma$ for $i \in 0, \ldots, C-1$, the total variation distance when including up until q_{C-1} becomes $\frac{1}{2}C\sigma$. For q_C we need to do something different as $q_c = 1 - \sum_0^{C-1} q_i = 1 - \sum_0^{C-1} p_i + X_i\sigma$, where X_i is equal to 1 or -1 with equal probability and independent for each i. Then

$$|p_c - q_c| = \left| 1 - \sum_0^{C-1} p_i - 1 + \sum_0^{C-1} p_i + X_i\sigma \right| = \sigma \left| \sum_0^{C-1} X_i \right|$$

This depends on the outcomes of the X_i, so instead we will use the expected value of $|p_c - q_c|$. We note that $X_i = 2Z_i - 1$ where Z_i is a Bernoulli random variable with success probability $\frac{1}{2}$. Then $\sum_{i=0}^{C-1} X_i = \sum_{i=0}^{C-1}(2Z_i - 1) = 2\sum_{i=0}^{C-1} Z_i - C$. Define $U = \sum_{i=0}^{C-1} Z_i$. Then U is Binomial$(C, \frac{1}{2})$ distributed. This gives us that

$$\mathbb{E}[|p_c - q_c|] = \mathbb{E}[\sigma\,|2U - C|] = \sigma \sum_{j=0}^{C} |2i - C|\,\mathbb{P}(U = i) = \sigma \sum_{j=0}^{C} \binom{C}{i} \frac{|2i - C|}{2^C} \tag{8.3}$$

All this combined means that $\mathbb{E}[\delta(\mathbb{P}, \mathbb{Q})] = \frac{\sigma}{2}\left(C + \sum_{j=0}^{C} \binom{C}{i} \frac{|2i-C|}{2^C}\right)$. So to ensure that in expectation $\delta(\mathbb{P}, \mathbb{Q}) = \sigma$, contestants should instead predict $q_i = p_i \pm \frac{2\sigma}{C + \sum_{j=0}^{C} \binom{C}{i} \frac{|2i-C|}{2^C}}$. Values of the factor $\frac{C + \sum_{j=0}^{C} \binom{C}{i} \frac{|2i-C|}{2^C}}{2}$ for the category numbers 2 through 8 are presented in Table 8.1

Number of categories $(C+1)$	2	3	4	5	6	7	8
Factor	3	4	5.5	6.5	6.875	7.875	8.875

Table 8.1: Value of the factor $\frac{C + \sum_{j=0}^{C} \binom{C}{i} \frac{|2i-C|}{2^C}}{2}$ for 2 through 8 categories

However, there are multiple ways to measure the distance between probability distributions. We choose total variation distance here because it is straightforward to compute for multi-categorical distributions, it does not depend on the values of p_i, and in the binary case, the distance simplifies to exactly σ. Nonetheless, it may also be useful to consider other divergence measures. The Kullback–Leibler divergence is another commonly used statistical distance.

Definition 8.2 (Kullback-Leibler divergence)**.** The Kullback-Leibler divergence is defined as:

$$D_{KL}(\mathbb{P}||\mathbb{Q}) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)}\, \mathrm{d}x$$

where $p(x)$ and $q(x)$ are the probability density functions of $\mathbb{P}$ and $\mathbb{Q}$. (Kullback & Leibler, 1951)

In the binary set-up, the Kullback–Leibler divergence becomes

$$D_{KL}(\mathbb{P}||\mathbb{Q}) = (1-p) \log \frac{1-p}{1-p \mp \sigma} + p \log \frac{p}{p \pm \sigma}$$

We can see that the D_{KL} depends on the success probability p and whether $+\sigma$ or $-\sigma$ was chosen. Also, if $p \pm \sigma = 1$ or 0, the Kullback-Leibler divergence takes the value infinity, unless p is equal to 1 or 0 as well (Csiszár, 1975). So this is not as suitable a choice as the total variation distance.

Another statistical distance we would like to consider is the Wasserstein distance. The Wasserstein distance was first defined by Leonid Kantorovich in 1960. However, it was named after Leonid Vaserstein, who used the metric in his work on Markov processes describing large systems of automata (Vaserstein, 1969).

Definition 8.3 (Wasserstein distance)**.** Let (M, d) be a metric space that is a Polish space. For $p \in [1, \infty]$ and two probability measure $\mathbb{P}$ and $\mathbb{Q}$ with finite p-moments, the Wasserstein p-distance is

$$W_p(\mathbb{P}, \mathbb{Q}) = \inf_{\gamma \in \Gamma(\mathbb{P}, \mathbb{Q})} (\mathbb{E}_{(x,y)\sim\gamma} d(x, y)^p)^{\frac{1}{p}}$$

where $\Gamma(\mathbb{P}, \mathbb{Q})$ is the set of all couplings of $\mathbb{P}$ and $\mathbb{Q}$. W_∞ is defined as $\lim_{p\to\infty} W_p(\mathbb{P}, \mathbb{Q})$. A coupling γ is a joint probability measure on $M \times M$, such that for all measurable $A \subset M$, $\gamma(A \times M) = \mathbb{P}(A)$ and $\gamma(M \times A) = \mathbb{Q}(A)$ (Kantorovich, 1960)

The Wasserstein distance is also known in computer science as the earth mover's distance. This is because, intuitively, if each distribution is viewed as a unit amount of earth on a metric space M, the Wasserstein distance is the minimum cost of turning one pile into the other. The cost is calculated as the amount of earth that needs to be moved times the mean distance it needs to be moved.

In 1974, Vallender showed that for probability measures on $\mathbb{R}$ with cumulative distribution functions $F_\mathbb{P}(x)$ and $F_\mathbb{Q}(x)$ and $p = 1$, the Wasserstein distance is equal to

$$W_1(\mathbb{P}, \mathbb{Q}) = \int_{-\infty}^{\infty} |F_\mathbb{P}(x) - F_\mathbb{Q}(x)| \, \mathrm{d}x \tag{8.4}$$

(Vallender, 1974) Because Definition 8.3 is not easy to compute, we will only consider the case $p = 1$ using Equation (8.4).

If we evaluate this for our multi-category setup, we get:

$$\begin{aligned}
W_1(\mathbb{P}, \mathbb{Q}) &= \int_{-\infty}^{\infty} |F_\mathbb{P}(x) - F_\mathbb{Q}(x)| \, \mathrm{d}x \\
&= \int_{-\infty}^{0} |F_\mathbb{P}(x) - F_\mathbb{Q}(x)| \, \mathrm{d}x + \sum_{i=0}^{C-1} \int_i^{i+1} |F_\mathbb{P}(x) - F_\mathbb{Q}(x)| \, \mathrm{d}x + \int_C^{\infty} |F_\mathbb{P}(x) - F_\mathbb{Q}(x)| \, \mathrm{d}x \\
&= \sum_{i=0}^{C-1} \int_i^{i+1} |F_\mathbb{P}(x) - F_\mathbb{Q}(x)| \, \mathrm{d}x \\
&= \sum_{i=0}^{C-1} 1 \cdot |F_\mathbb{P}(i) - F_\mathbb{Q}(i)| \\
&= \sum_{i=0}^{C-1} \left| \sum_{j=0}^{i} p_j - \sum_{j=0}^{i} q_j \right|
\end{aligned}$$

In our case with binary random variables, this simplifies to $|(1-p) - (1-p\pm\sigma)| = \sigma$. This is the same as we saw for the total variation distance. However, when we look at multiple categories, a problem starts to arise. Let X_i be the sign of σ for the i-th event when contestants predict $p_i \pm \sigma$. So $X_i = \pm 1$ each with probability $\frac{1}{2}$. Then $W_1(\mathbb{P}, \mathbb{Q})$ works out to:

$$W_1(\mathbb{P}, \mathbb{Q}) = \sigma \sum_{i=0}^{C-1} \left| \sum_{j=0}^{i} X_j \right|$$

So, the Wasserstein distance depends on the values of X_i. In the case of the total variation distance, this was only the case for p_C. Therefore, the Wasserstein distance is not as favorable a choice as the total variation distance, but, like we did for the total variation distance, it might still be salvageable by considering the expected distance.

Let $Y_i = \sum_{j=0}^{i} X_j$. Then

$$\begin{aligned}
\mathbb{E}[W_1(\mathbb{P}, \mathbb{Q})] &= \mathbb{E}\left[\sigma \sum_{i=0}^{C-1} |Y_i|\right] \\
&= \sigma \sum_{i=0}^{C-1} \mathbb{E}[|Y_i|]
\end{aligned} \tag{8.5}$$

So we seek to compute $\mathbb{E}[|Y_i|]$. First, as noted before, $X_j = 2Z_j - 1$ where Z_j is a Bernoulli random variable with success probability $\frac{1}{2}$. Then $Y_i = \sum_{j=0}^{i} X_j = \sum_{j=0}^{i}(2Z_j - 1) = 2\sum_{j=0}^{i} Z_j - (i+1)$. Define $U_i = \sum_{j=0}^{i} Z_j$. Then U_i is Binomial$(i+1, \frac{1}{2})$ distributed. This gives us that

$$\mathbb{E}[|Y_i|] = \mathbb{E}[|2U_i - (i+1)|] = \sum_{j=0}^{i+1} |2j - (i+1)|\,\mathbb{P}(U_i = j) = \sum_{j=0}^{i+1} \binom{i+1}{j} \frac{|2j-(i+1)|}{2^{i+1}} \tag{8.6}$$

Combining equation 8.5 and 8.6 yields that $\mathbb{E}[W_1(\mathbb{P}, \mathbb{Q})] = \sigma \sum_{i=0}^{C-1} \sum_{j=0}^{i+1} \binom{i+1}{j} \frac{|2j-(i+1)|}{2^{i+1}}$. So to get an expected distance of σ, we should let contestants predict $p_i \pm \frac{\sigma}{\sum_{i=0}^{C-1} \sum_{j=0}^{i+1} \binom{i+1}{j} \frac{|2j-(i+1)|}{2^{i+1}}}$. Table 8.2 shows the values of this factor $\sum_{i=0}^{C-1} \sum_{j=0}^{i+1} \binom{i+1}{j} \frac{|2j-(i+1)|}{2^{i+1}}$ for categories 2 through 8.

Number of categories $(C+1)$	2	3	4	5	6	7	8
Factor	1	2	3.5	5	6.875	8.75	10.9375

Table 8.2: Value of the factor $\sum_{i=0}^{C-1} \sum_{j=0}^{i+1} \binom{i+1}{j} \frac{|2j-(i+1)|}{2^{i+1}}$ for 2 through 8 categories

Figures 8.2 and 8.3 show the distribution of the ranks of the winner in tournaments with events having 3, 4, 5, 6, 7, and 8 categories, using the Multi-Category Brier Score, with the total variation distance and the Wasserstein distance, respectively. In Section 8.3, we will discuss the use of other scoring rules.

In Figure 8.2, we can see that the distribution of winners remains largely the same across all category counts when using the total variation distance. The distribution is also nearly identical to that in the binary setting shown in Figure 2.1. In Figure 8.3, the distribution of ranks of winners shifts gradually to the right as the number of categories increases. It also does not resemble the distribution for binary events shown in Figure 2.1. This suggests that the Wasserstein distance is not a good choice of distance, and that the total variation distance behaves much more consistently. This is also the distance we will use from now on.

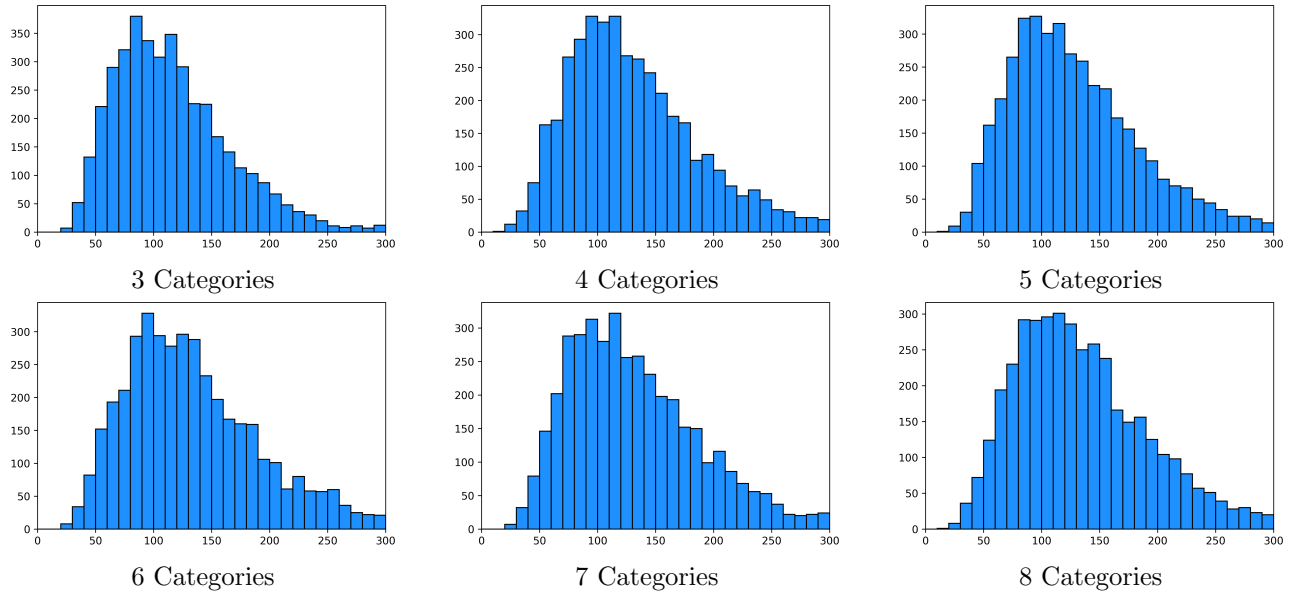


Figure 8.2: Rank tournament winner for 3, 4, 5, 5, 6, 7 and 8 categories, total variation distance, Multi-Category Brier Score, $0 < \sigma < 0.3$

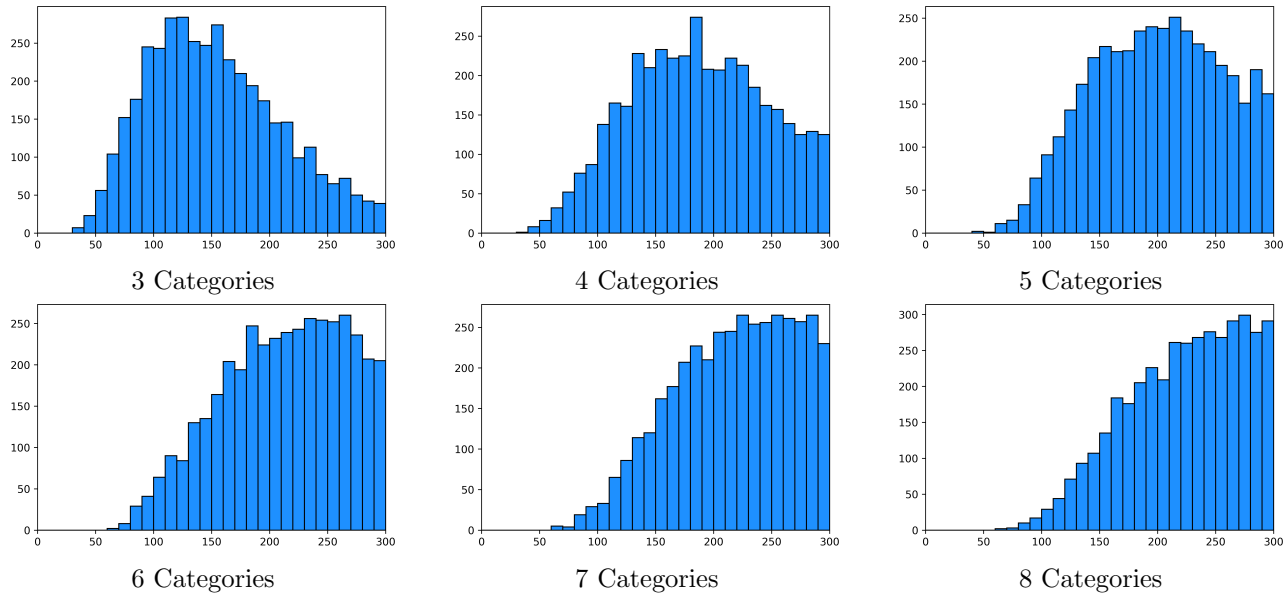


Figure 8.3: Rank tournament winner for 3, 4, 5, 5, 6, 7 and 8 categories, Wasserstein distance, Multi-Category Brier Score, $0 < \sigma < 0.3$

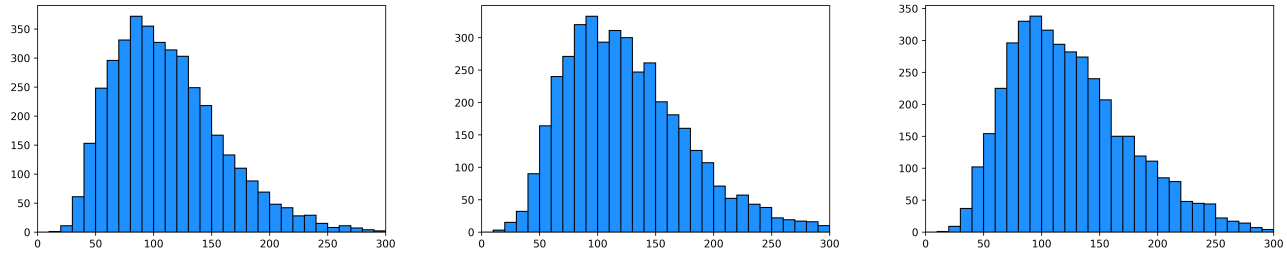
8.3 Scoring Rules

In Section 7, we compared the Brier, Logarithmic, Spherical, Pseudospherical, and Power scores for binary events. The latter two were evaluated for several values of the parameter β, and we found that smaller values of β yielded better results. In this section, we analyze the same scoring rules for multi-category events. For the Pseudospherical and Power scores, we consider only the parameter value $\beta = 1.01$, as this provided good performance in the binary case. Additionally, we introduce a new scoring rule: the Continuous Ranked Probability Score (CRPS), presented in Example 3.9. Equation (3.14) provides the CRPS formulation for the multi-category case. We did not use the CRPS for binary events, as it is then equivalent to the Brier score. The CRPS is symmetric.

The distribution of winner ranks for each scoring rule (except the Brier score, whose results are shown in Figure 8.2) for 3, 4, and 5 categories and $0 < \sigma < 0.3$ is shown in Figures 8.4 through 8.8. Across all scoring rules, we observe a relatively consistent distribution shape as the number of categories increases.

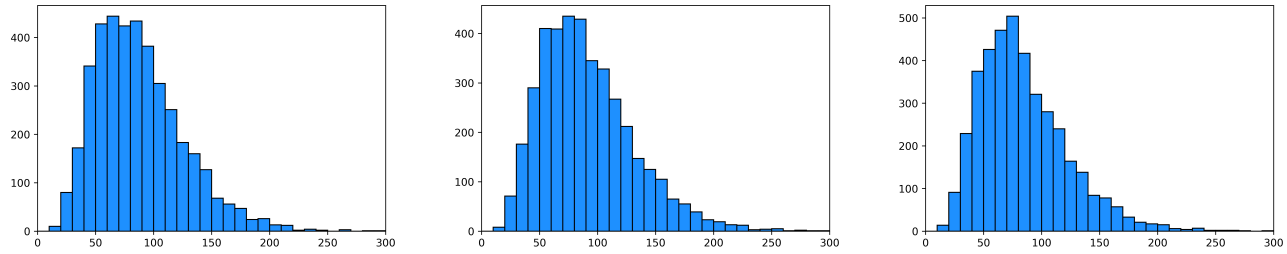
Figure 8.4, in which the CRPS is used, shows a distribution of winner ranks similar to that of the Brier score in Figure 8.2. Contestants around rank 100 are most likely to win the tournament. The Spherical score, shown in Figure 8.6, produces comparable results for 3 categories. For 4 and 5 categories, however, the rank of the contestant most likely to win shifts slightly from around 100 to approximately 110. This shift is relatively small, so we cannot conclude that the Spherical score performs significantly worse than the CRPS or Brier score.

However, we observe a clear improvement with the Logarithmic, and Power and Pseudospherical scores using $\beta = 1.01$, as shown in Figures 8.5 (Logarithmic score), 8.7 (Power score) and 8.8 (Pseudospherical score). In all three cases cases, the contestant most likely to win the tournament is ranked around 70. Additionally, contestants ranked above 200 rarely win, whereas with the other scoring rules this occurs more frequently. Taken together, these observations suggest that the Power and Pseudospherical scores with $\beta = 1.01$ and the Logarithmic score are more suitable choices than the Brier, CRPS, or Spherical scores.



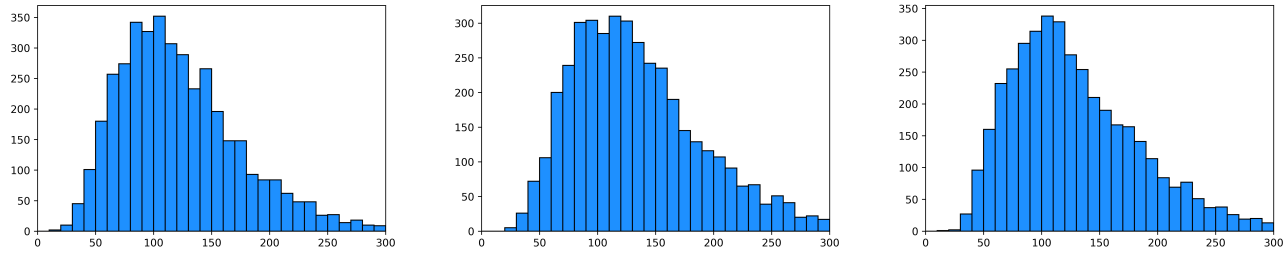
3 Categories 4 Categories 5 Categories

Figure 8.4: Rank tournament winner for 3, 4, and 5 categories, total variation distance, Continuous Ranked Probability Score, $0 < \sigma < 0.3$



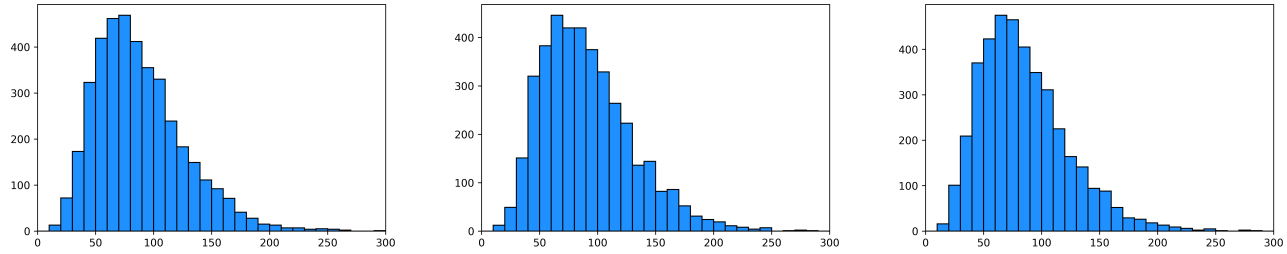
3 Categories 4 Categories 5 Categories

Figure 8.5: Rank tournament winner for 3, 4, and 5 categories, total variation distance, Logarithmic Score, $0 < \sigma < 0.3$



3 Categories 4 Categories 5 Categories

Figure 8.6: Rank tournament winner for 3, 4, and 5 categories, total variation distance, Spherical Score, $0 < \sigma < 0.3$



3 Categories 4 Categories 5 Categories

Figure 8.7: Rank tournament winner for 3, 4, and 5 categories, total variation distance, Power Score $\beta = 1.01$, $0 < \sigma < 0.3$

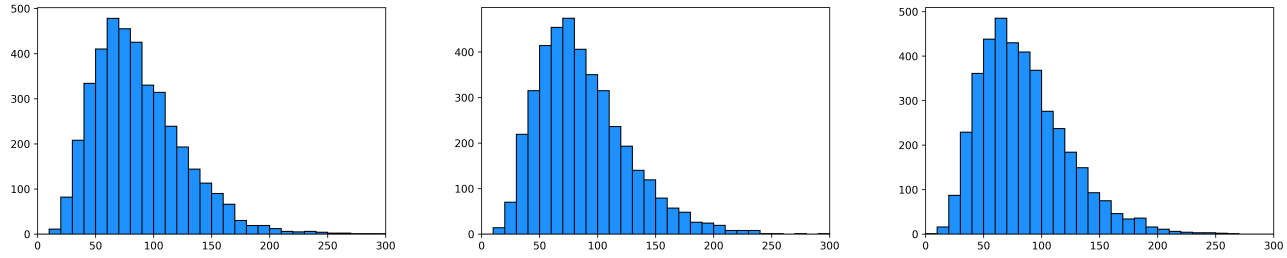


Figure 8.8: Rank tournament winner for 3, 4, and 5 categories, total variation distance, Pseudospherical Score $\beta = 1.01$, $0 < \sigma < 0.3$

8.4 Conclusions

In this section, we extended our framework from binary events to multi-category random variables. To achieve this, we introduced a new sampling strategy for generating the true probabilities for each category, as described in Section 8.1. In Section 8.2, we examined how to model the predictions of contestants, considering several statistical distance measures: total variation distance, Kullback–Leibler divergence, and Wasserstein distance. After careful evaluation, we selected the total variation distance as the most suitable choice for our simulations.

Next, in Section 8.3, we analyzed the performance of different scoring rules. The Brier score and CRPS both yielded similar results. The Spherical score performed slightly worse for 4 and 5 categories, though the difference was not substantial. Notably, we observed improved outcomes with the Power and Pseudospherical scores using $\beta = 1.01$ and the Logarihtmic score: the rank of the contestant most likely to win the tournament improved from around 100 to approximately 70. Based on these findings, we recommend using the Power or Pseudospherical score with $\beta = 1.01$ or Logarithmic score for tournaments involving multi-category events.

9 Continuous random variables

In Section 8, we expanded our simulation to include events with multiple categories. We now aim to extend this further to continuous random variables, allowing for an even broader range of possible prediction competitions. As in Section 8, we will use the total variation distance to evaluate contestants' predictions. In this section, we focus on on Normal distributions with known variance. First, we will discuss in Section 9.1 how to extend the model to accommodate normal distributions. Then we look at several scoring rules and compare their performance in the scenario of a normal distribution in Section 9.2.

9.1 Normal distribution

We move on to the case where the true distribution $\mathbb{D}$ is normally distributed and contestants predict a normal distribution as well. For simplicity, we only look at the case where the variance τ^2 of $\mathbb{D}$ is known, so contestants predict a normal distribution with the same variance τ^2 but with varying mean μ. As the Normal distribution is symmetric and changing its mean only causes a horizontal shift of the probability density function, it does not matter what the true mean of $\mathbb{D}$, so we set it to 0. We model contestants predictions in the following way: they predict a normal distribution $\mathbb{Q}_1$ or $\mathbb{Q}_2$, each with equal probability, where μ_1 and μ_2 of $\mathbb{Q}_1$ and $\mathbb{Q}_2$ is chosen in such a way that the total variation distance between $\mathbb{D}$ and both $\mathbb{Q}_1$ and $\mathbb{Q}_2$ is σ. Because of symmetry of both the normal distribution and the total variation distance, if we find μ_1 for $\mathbb{Q}_1$, we know that $\mu_2 = -\mu_1$.

Now to compute μ we first compute the total variation distance between two normal distributions X and Y. Where X is $\mathcal{N}(0, \tau^2)$ distributed and Y is $\mathcal{N}(\mu, \tau^2)$ distributed. Note that $X = \tau N$ and $Y = \tau N + \mu$ where N is $\mathcal{N}(0, 1)$ distributed.

$$\begin{aligned}\delta(X, Y) &= \frac{1}{2}\int |f_X(x) - f_Y(x)| \, \mathrm{d}x \\ &= \frac{1}{2}\left(\int_{-\infty}^{\infty} \left| \frac{1}{\sqrt{2\pi\tau^2}} \mathrm{e}^{-\frac{x^2}{2\tau^2}} - \frac{1}{\sqrt{2\pi\tau^2}} \mathrm{e}^{-\frac{(x-\mu)^2}{2\tau^2}} \right| \, \mathrm{d}x\right) \end{aligned} \tag{9.1}$$

To determine when the absolute sign take positive value and when negative we compute the following:

$$\begin{aligned}\frac{1}{\sqrt{2\pi\tau^2}} \mathrm{e}^{-\frac{x^2}{2\tau^2}} &= \frac{1}{\sqrt{2\pi\tau^2}} \mathrm{e}^{-\frac{(x-\mu)^2}{2\tau^2}} \\ -\frac{x^2}{2\tau^2} &= -\frac{(x-\mu)^2}{2\tau^2} \\ x^2 &= (x-\mu)^2 \\ x^2 &= x^2 - 2x\mu + \mu^2 \\ 2x\mu &= \mu^2 \\ x &= \frac{\mu}{2}\end{aligned}$$

So we can rewrite equation (9.1) to

$$\begin{aligned}\delta(X, Y) &= \frac{1}{2}\left(\int_{-\infty}^{\frac{\mu}{2}} \frac{1}{\sqrt{2\pi\tau^2}} \mathrm{e}^{-\frac{x^2}{2\tau^2}} - \frac{1}{\sqrt{2\pi\tau^2}} \mathrm{e}^{-\frac{(x-\mu)^2}{2\tau^2}} \, \mathrm{d}x + \int_{\frac{\mu}{2}}^{\infty} -\frac{1}{\sqrt{2\pi\tau^2}} \mathrm{e}^{-\frac{x^2}{2\tau^2}} + \frac{1}{\sqrt{2\pi\tau^2}} \mathrm{e}^{-\frac{(x-\mu)^2}{2\tau^2}} \, \mathrm{d}x\right) \\ &= \frac{1}{2}\left(\mathbb{P}\left(X \le \frac{\mu}{2}\right) - \mathbb{P}\left(Y \le \frac{\mu}{2}\right) - \mathbb{P}\left(X \ge \frac{\mu}{2}\right) + \mathbb{P}\left(Y \ge \frac{\mu}{2}\right)\right) \\ &= \frac{1}{2}\left(\mathbb{P}\left(X \le \frac{\mu}{2}\right) - \mathbb{P}\left(Y \le \frac{\mu}{2}\right) - 1 + \mathbb{P}\left(X \le \frac{\mu}{2}\right) + 1 - \mathbb{P}\left(Y \le \frac{\mu}{2}\right)\right) \\ &= \frac{1}{2}\left(2\mathbb{P}\left(X \le \frac{\mu}{2}\right) - 2\mathbb{P}\left(Y \le \frac{\mu}{2}\right)\right) \\ &= \mathbb{P}\left(\tau N \le \frac{\mu}{2}\right) - \mathbb{P}\left(\tau N + \mu \le \frac{\mu}{2}\right) \\ &= \mathbb{P}\left(N \le \frac{\mu}{2\tau}\right) - \mathbb{P}\left(N \le -\frac{\mu}{2\tau}\right)\end{aligned}$$

$$= \mathbb{P}\left(N \leq \frac{\mu}{2\tau}\right) - \mathbb{P}\left(N \leq -\frac{\mu}{2\tau}\right) = \mathbb{P}\left(-\frac{\mu}{2\tau} \leq N \leq \frac{\mu}{2\tau}\right)$$

Because of symmetry around 0, $\mathbb{P}\left(-\frac{\mu}{2\tau} \leq N \leq \frac{\mu}{2\tau}\right) = 2\left(\mathbb{P}\left(N \leq \frac{\mu}{2\tau}\right) - \frac{1}{2}\right) = 2\Phi\left(\frac{\mu}{2\tau}\right) - 1$.

So to achieve a distance of σ, we need that

$$\begin{aligned}
\sigma &= 2\Phi\left(\frac{\mu}{2\tau}\right) - 1 \\
\Phi\left(\frac{\mu}{2\tau}\right) &= \frac{\sigma+1}{2} \\
\frac{\mu}{2\tau} &= \Phi^{-1}\left(\frac{\sigma+1}{2}\right) \\
\mu &= 2\tau\Phi^{-1}\left(\frac{\sigma+1}{2}\right)
\end{aligned}$$

So Q_1 is $\mathcal{N}(2\tau\Phi^{-1}\left(\frac{\sigma+1}{2}\right), \tau^2)$ distributed and $\mathbb{Q}_2$ is $\mathcal{N}(-2\tau\Phi^{-1}\left(\frac{\sigma+1}{2}\right), \tau^2)$ distributed.

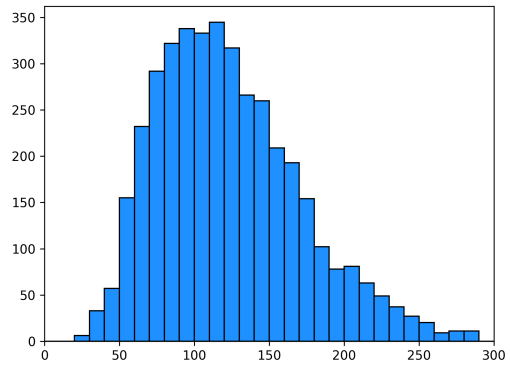
Now that we have determined what distributions contestants should predict, we move on to the scoring rule. For now we use the Quadratic score, a continuous extension of the Brier score, as scoring rule. In Section 9.2, we discuss various other scoring rules that could be used in this scenario and compare results.

Example 9.1 (Quadratic Score)**.** For a Binary outcome $y \in \{0, 1\}$, the Quadratic score is

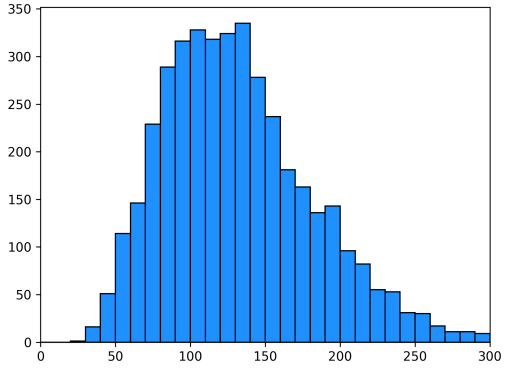
$$S_{\text{Quadratic}}(\mathbb{P}, y) = -2f_{\mathbb{P}}(y) + \int_{-\infty}^{\infty} f_{\mathbb{P}}(x)^2 \, \mathrm{d}x$$

where $f_{\mathbb{P}}(x)$ is the predicted value for the event $\{Y = 1\}$, is a strictly proper scoring rule. (Brier, 1950)

Figure 9.1 shows the distribution of ranks of the winner for τ^2 between 0.5 and 1.5 and between 0.5 and 10.5. We take 100 different values for τ^2 equally spread over their respective interval. The distributions shown in Figure 9.1 have a similar shape to Figure 2.1 for the binary case arise. The winner is likely to come from around the 100th most accurate contestant for τ^2 between 0.5 and 1.5 and the 110th most accurate contestant for τ^2 between 0.5 and 10.5. Other than this, we observe little difference between τ^2 in range [0.5, 1.5] and τ^2 in range [0.5, 10.5].



$0.5 < \tau^2 < 1.5$



$0.5 < \tau^2 < 10.5$

Figure 9.1: Rank tournament winner with true distribution $\mathcal{N}(0, \tau^2)$ with $0.5 < \tau^2 < 1.5$ and $0.5 < \tau^2 < 10.5$, total variation distance, Quadratic Score, $0 < \sigma < 0.3$

9.2 Scoring Rules

We now want to analyze the effect that different scoring rules have on the distribution of winner ranks in the setting of Normal distributions. The proper scoring rules introduced in section 3 for discrete random variables

can also be used for continuous random variables by changing predicted probabilities to predicted densities and summation over the entire domain to integration over the entire domain. The following is an overview of the continuous version of all the proper scoring rules we would like to compare. S is the assigned score, $\mathbb{Q}$ the predicted distribution, $p(x)$ the corresponding density, $F_{\mathbb{Q}}$ the corresponding cdf and y the outcome.

1. **Quadratic Score:**
$$S_{\text{Quadratic}}(\mathbb{Q}, y) = -2p(y) + \int_{-\infty}^{\infty} p(x)^2 \, \mathrm{d}x, \tag{9.2}$$

2. **Logarithmic Score:**
$$S_{\text{Log}}(\mathbb{Q}, y) = -\log(p(y)) \tag{9.3}$$

3. **Spherical Score:**
$$S_{\text{Sphere}}(\mathbb{Q}, y) = -\frac{p(y)}{\sqrt{\int_{-\infty}^{\infty} p(x)^2 \mathrm{d}x}} \tag{9.4}$$

4. **Pseudospherical Score:**
$$S_{\text{Pseudosphere}}(\mathbb{Q}, y) = -\left(\frac{p(y)}{\sqrt[\beta]{\int_{-\infty}^{\infty} p(x)^\beta \mathrm{d}x}} \right)^{\beta-1} \tag{9.5}$$

5. **Power Score:**
$$S_{\text{Power}}(\mathbb{Q}, y) = -\beta p(y)^{\beta-1} + (\beta - 1) \int_{-\infty}^{\infty} p(x)^\beta \mathrm{d}x \tag{9.6}$$

6. **Continuous Ranked Probability Score:**
$$S_{\text{CRPS}}(\mathbb{Q}, y) = \int_{-\infty}^{\infty} (F_{\mathbb{Q}}(x) - \mathbf{1}\{y \leq x\})^2 \, \mathrm{d}x \tag{9.7}$$

The Continuous Ranked Probability Score when $\mathbb{Q}$ is a $\mathcal{N}(\mu, \tau^2)$ distribution evaluates to the following

$$S_{\text{CRPS}}(\mathbb{Q}, y) = \tau \left(\frac{y-\mu}{\tau} \left[2\Phi\left(\frac{y-\mu}{\tau} \right) - 1 \right] + 2\phi\left(\frac{y-\mu}{\tau} \right) - \frac{1}{\sqrt{\pi}} \right) \tag{9.8}$$

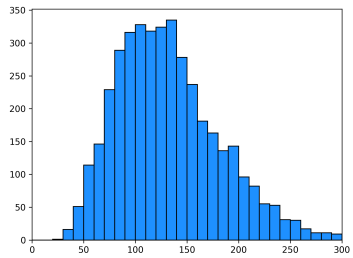
(Gneiting, Raftery, Westveld III, & Goldman, 2005)

To compute the value of the other scoring rules we need to evaluate $p(y)$ and $\int_{-\infty}^{\infty} p(x)^\beta \mathrm{d}x$ ($\beta = 2$ for the quadratic and spherical score). $p(y)$ is equal to $\frac{1}{\sqrt{2\pi\tau^2}} \mathrm{e}^{-\frac{(y-\mu)^2}{2\tau^2}}$ and $\int_{-\infty}^{\infty} p(x)^\beta \mathrm{d}x$ evaluates to:

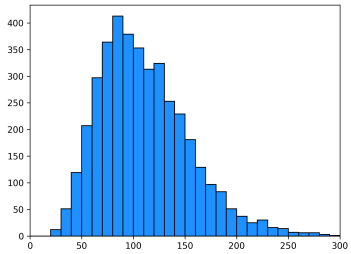
$$\begin{aligned}
\int_{-\infty}^{\infty} p(x)^\beta \mathrm{d}x &= \int_{-\infty}^{\infty} \left(\frac{1}{\sqrt{2\pi\tau^2}} \mathrm{e}^{-\frac{(x-\mu)^2}{2\tau^2}} \right)^\beta \mathrm{d}x \\
&= \frac{1}{\sqrt{2\pi\tau^2}^\beta} \int_{-\infty}^{\infty} \mathrm{e}^{-\beta\frac{(x-\mu)^2}{2\tau^2}} \mathrm{d}x \\
&= \frac{1}{\sqrt{2\pi\tau^2}^\beta} \int_{-\infty}^{\infty} \mathrm{e}^{-\frac{(x-\mu)^2}{2\left(\frac{\tau}{\sqrt{\beta}}\right)^2}} \mathrm{d}x \\
&= \frac{1}{\sqrt{2\pi\tau^2}^\beta} \sqrt{2\pi \left(\frac{\tau}{\sqrt{\beta}} \right)^2} \\
&= \frac{1}{\sqrt{2\pi\tau^2}^\beta} \sqrt{2\pi\tau^2} \frac{1}{\sqrt{\beta}} \\
&= \frac{1}{\sqrt{\beta}} \left(\frac{1}{\sqrt{2\pi\tau^2}} \right)^{\beta-1}
\end{aligned}$$

Figure 9.2 shows the distribution of ranks of the winner for the mentioned scoring rules. We only consider the case where τ^2 is in range [0.5, 10.5] as the two ranges produced similar results for the Quadratic score in Figure 9.1. We can observe several differences and similarities between the different scores.

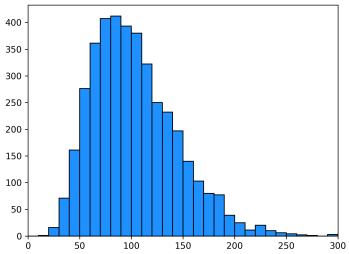
We observe a similar distribution of ranks for the Quadratic score, CRPS and Spherical score. The peak of the distribution is around rank 100 for all these scores. The Logarithmic score, Power score and Pseudospherical score with $\beta = 1.01$ also perform similarly, but they perform notably better than the other three scoring rules. The contestant most likely to win the tournament shifts from around rank 100 to around rank 85. So when setting up a tournament with random variables that have a Normal distribution with known variance, the Logarithmic score, Power score and Pseudospherical score with $\beta = 1.01$ are preferred.



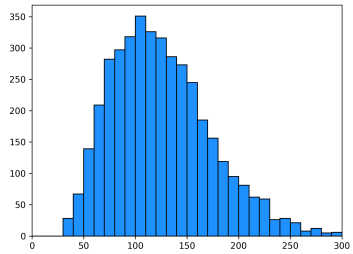
Quadratic score



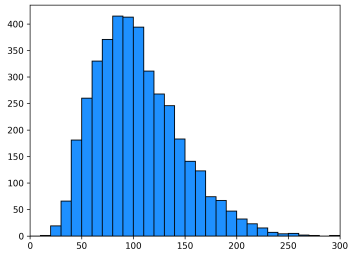
CRPS



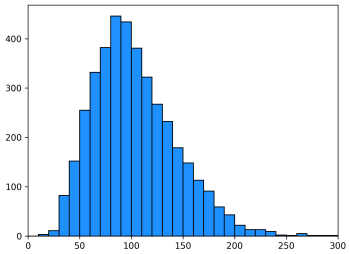
Logarithmic score



Spherical score



Power score $\beta = 1.01$



Pseudospherical score $\beta = 1.01$

Figure 9.2: Rank tournament winner with true distribution $\mathcal{N}(0, \tau^2)$ with $0.5 < \tau^2 < 10.5$, total variation distance, several different scoring rules, $0 < \sigma < 0.3$

9.3 Conclusions

In Section 9.1, we discussed how to extend our model to accommodate random variables with a normal distribution with known variance. We again used the total variation distance to model contestant predictions. We showed results for known variance in ranges [0.5, 1.5] and [0.5, 10.5] and found little difference in distribution between the two other than that the first has its peak at around 100 and the second at around 110.

Then, in Section 9.2 we compared the distribution of ranks of the winner for several different scoring rules and found that the Power and Pseudospherical score with $\beta = 1.01$, and the Logarithmic Score performed the best.

10 Conclusion

We began by analyzing the paradox described in Section 2 in more detail in Section 4, focusing on the finish positions and scores of the contestants. We observed that the average finish position increased with contestant rank for σ in the ranges [0, 0.3], [0.05, 0.35], [0.1, 0.4], and [0.15, 0.45]. Specifically, for σ in the range [0, 0.3], the most accurate predictor was most likely to finish in 35th place. We also analyzed the average rank per finish position. For the same σ range, we found that the lowest average rank occurred at finish position 35. When focusing on the ranks at finish position 35, rather than on the winner of the tournament, we observed significantly better results. Applying the same technique for σ between 0.05 and 0.35 also yielded improved results, although the improvement was less pronounced.

In addition to finish positions, we also examined the scores of contestants for σ between 0 and 0.3. The average score increases with contestant rank, and the range of scores also broadens as rank increases. We compared the scores of the tournament winner to those of the most accurate predictor and found that both have similar score distributions. However, the distribution for the most accurate predictor is shifted approximately one point to the right relative to that of the winner. When analyzing the distribution of score differences between the winner and the most accurate predictor, we found that a difference of 1 point is the most common. Furthermore, while higher-ranked contestants rarely outperform the most accurate predictor, when they do, they tend to win by a significantly larger margin than lower-ranked contestants.

We analyzed the top 5, 10, and 20 finishers for $0 < \sigma < 0.3$. While the distribution of ranks among these top finishers resembles that of the tournament winner, it shifts progressively to the left as more finishers are included. Notably, the 10 most accurate contestants never appear in the top 5 and only rarely appear in the top 10 or top 20.

Next, we adjusted the success probabilities as described in Section 5. We found that when success probabilities were skewed toward the lower and upper ends of the interval [0, 1], the rank of the contestant most likely to win decreased from 100 to around 75. Although the overall shape of the distribution remained similar, it shifted noticeably to the left. Conversely, when the success probabilities were concentrated near the center of the interval [0, 1], the rank of the most likely winner increased to 115, and the distribution of winners' ranks shifted to the right. Sampling success probabilities from beta distributions (with parameters $\alpha = \beta = 0.2, 1, 5$) produced effects similar to those observed when selecting success probabilities manually.

Then, in Section 6, we explored the impact of letting contestants predict values from continuous distributions, as opposed to using the original predict $p \pm \sigma$ approach. In Section 6.1, we examined contestants making predictions drawn uniformly from the interval $[p - \sigma\sqrt{3}, p + \sigma\sqrt{3}]$. Section 6.2 extended this analysis to various Beta distributions (both symmetric and asymmetric). In both cases, when the variance is fixed at σ^2, the distribution of winners remains largely unchanged compared to the $p \pm \sigma$ strategy. In Section 6.3, we considered a case where contestants make extreme predictions (always choosing 0 or 1). This approach resulted in a more dispersed distribution of winners across the ranks, with higher-ranked contestants having a greater probability of winning. However, in a mixed tournament, the extreme prediction strategy proved to be less effective. Contestants using the predict $p \pm \sigma$ method were significantly more likely to win.

In Section 7, we observed that the choice of scoring rule has an impact. The Logarithmic scoring rule yielded better results compared to the Spherical and Brier Score, when evaluating the tournament winner but performed worse in terms of the average finishing position for lower-ranked contestants. Moreover, the advantage of the Logarithmic score diminished when the true probabilities were concentrated toward the center of the interval [0, 1].

We also found that the Pseudospherical score did not outperform the Spherical score for $\beta > 2$. As β approached 1, performance improved slightly, but the gains were not substantial. Similarly, the Power score performed worse for larger values of β, while a significant improvement was observed when β was closer to 1. However, as with the Logarithmic score, this improvement disappeared when the true probabilities were concentrated near the middle of the interval [0, 1].

In Section 8, we extended our framework from binary events to multi-category random variables. To achieve this, we introduced a new sampling strategy for generating the true probabilities for each categor and examined which probabilities contestants should predict with the use of a statistical distance. We selected total variation distance as the most suitable choice for our simulations. We then analyzed the performance of different scoring rules. The Brier, CRPS, and Logarithmic scores all yielded similar results to the binary case. The Spherical score performed slightly worse, though the difference was not substantial. Notably, we observed improved outcomes with the Power and Pseudospherical scores using $\beta = 1.01$: the rank of the contestant most likely to

win the tournament improved from around 100 to approximately 70. Based on these findings, we recommend using the Power or Pseudospherical score with $\beta = 1.01$ for tournaments involving multi-category events.

We then turned to continuous random variables, extending our model to accommodate random variables following a normal distribution with known variance. As before, we used the total variation distance to model contestant predictions. We presented results for known variances in the ranges $[0.5, 1.5]$ and $[0.5, 10.5]$, and found little difference in the overall distribution, except that the former peaked at around 100, while the latter peaked at around 110. We then compared the distribution of winner ranks across several different scoring rules and found that the Power and Pseudospherical scores with $\beta = 1.01$, as well as the Logarithmic Score, performed the best.

Due to time constraints, we limited our analysis in Section 9 on continuous random variables to normal distributions with known variance. A natural extension would be to consider normal distributions with a known mean and unknown variance, or cases where both the mean and variance are unknown. It would also be interesting to explore other distributions with known densities and analyze their impact on the paradox. The optimal scoring rule may vary depending on the distribution considered. Additionally, mixed tournaments, where contestant predictions and the true distribution come from different types of distributions rather than the same distribution with varying parameters, could offer further insight.

It is also important to acknowledge that our current model has several limitations. We assume that contestant predictions are independent across the 100 random variables considered in the tournament, which may not reflect reality. Contestants might be more likely to make similar predictions for certain groups of random variables. Additionally, contestants may also be biased in one direction for certain random variables and in the other direction for others. We also assume that the outcomes of the random variables themselves are independent, which may not always be the case. These factors are not accounted for in our current model. Addressing these limitations would be a valuable direction for future research.

But what do these results mean for organizations designing prediction tournaments? Regardless of whether the tournament involves binary, multi-category, or normal (with known variance) random variables, the choice of scoring rule plays an importatn role. We found that more accurate contestants were more likely to win when using the Power or Pseudospherical score with values of β slightly greater than 1, or the Logarithmic score. However, since the Logarithmic score can assign an infinite penalty, the Power and Pseudospherical scores are generally more preferred. In the case of binary random variables, the advantage of these scoring rules over the Brier score disappeared when the true success probabilities were near 0.5. In such cases, the choice of scoring rule becomes less important. We also explored the idea of awarding victory to a contestant other than the one in first place as a way to mitigate the paradox. While this approach showed some potential, it is highly sensitive to the standard deviation σ of contestants. Since σ is unknown in real-world tournaments, this method is not practical.

What is the takeaway for contestants? If a contestant knows they make highly accurate predictions, introducing a small amount of noise to their predictions can actually increase their chances of winning. However, this benefit only holds up to a certain threshold, beyond that point, additional noise begins to reduce their chances. For contestants who do not make accurate predictions, adding noise only worsens their performance. We also explored an alternative strategy where contestants always predict 0 or 1. If all contestants in a tournament adopt this strategy, contestants of higher rank tend to have an increased chance of winning. However, in a mixed tournament, where not all contestants predict in this extreme way, a contestant is better off not predicting extremely as this severely reduces their chance of winning. There may still be other, unexplored strategies that contestants could use to improve their chances. Investigating such strategies would be a interesting direction for further research.

Bibliography

Aldous, D. J. (2019). A prediction tournament paradox. *The American Statistician*, *75*(3), 243–248. Retrieved from https://doi.org/10.1080/00031305.2019.1604430

Brier, G. (1950). Verification of forecasts expressed in terms of probability. *Monthy Weather Review*, *78*, 1–3. Retrieved from https://doi.org/10.1175/1520-0493(1950)078%3C0001:VOFEIT%3E2.0.CO;2

Buchweitz, E., Romano, J., & Tibshirani, R. (2025). Asymmetric penalties underlie proper loss functions in probabilistic forecasting. ArXiv Preprint: 2505.00937. Retrieved from https://doi.org/10.48550/arXiv.2505.00937

Csiszár, I. (1975). I-divergence geometry of probability distributions and minimization problems. *The Annals of Probability*, *3*(1), 146–158. Retrieved from https://doi.org/10.1214/aop/1176996454

Gneiting, T., Raftery, A., Westveld III, A., & Goldman, T. (2005). Calibrated probabilistic forecasting using ensemble model output statistics and minimum crps estimation. *Monthly Weather Review*, *133*(5), 1098–1118. Retrieved from https://doi.org/10.1175/MWR2904.1

Good, I. (1952). Rational decisions. *Journal of the Royal Statistical Society: Series B*, *14*, 107–114. Retrieved from https://doi.org/10.1111/j.2517-6161.1952.tb00104.x

Jose, V. (2007). A characterization for the spherical scoring rule. *Theory and Decision*, *66*, 263–281. Retrieved from https://doi.org/10.1007/s11238-007-9067-x

Kantorovich, L. (1960). Mathematical methods of organizing and planning production. *Management Science*, *6*(4), 366–422. Retrieved from https://doi.org/10.1137/1118101

Kolmogorov, A. (1963). On the approximation of distributions of sums of independent summands by infinitely divisible distribution. *Sankhyā: The Indian Journal of Statistics, Series A*, *25*(2), 159–174. Retrieved from https://doi.org/10.1016/B978-1-4832-3160-0.50016-6

Kullback, S., & Leibler, R. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, *22*(1), 79–86. Retrieved from https://doi.org/10.1214/aoms/1177729694

Matheson, J., & Winkler, R. (1976). Scoring rules for continuous probability distributions. *Management Science*, *22*, 1087–1096. Retrieved from https://doi.org/10.1287/mnsc.22.10.1087

Roby, T. (1965). Belief states and the uses of evidence. *Behavioral Science*, *10*, 255–270. Retrieved from https://doi.org/10.1002/bs.3830100304

Selten, R. (1998). Axiomatic characterization of the quadratic scoring rule. *Experimental Economics*, *1*, 43–61. Retrieved from https://doi.org/10.1023/A:1009957816843

Shuford, E., Albert, A., & Massengill, H. E. (1966). Admissible probability measurement procedures. *Psychometrika*, *31*, 125–145. Retrieved from https://doi:10.1007/BF02289503

Tetlock, P., & Hardner, D. (2015). *Superforecasting: The art and science of prediction*. Crown Publishing Group.

Toda, M. (1963). Measurement of subjective probability distribution. *Division of Mathematical Psyychology, Institute for Research*.

Tsybakov, A. (2009). *Introduction to nonparametric estimation*. Springer.

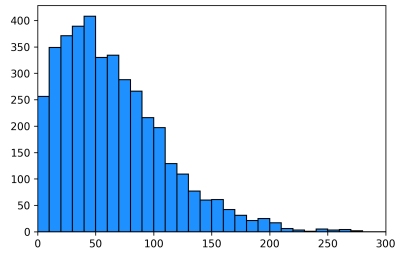
Vallender, S. (1974). Calculation of the wasserstein distance between probability distributions on the line. *Theory of Probability and Its Applications*, *18*(4), 784–786. Retrieved from https://doi.org/10.1137/1118101

Vaserstein, L. (1969). Markov processes over denumerable products of spaces, describing large systems of automata. *Problemy Peredachi Informatsii, 5*(3), 64–72.

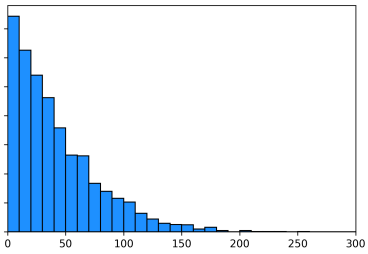
Waghmare, K., & Ziegel, J. (2025). Proper scoring rules for estimation and forecast evaluation. ArXiv Preprint: 2504.01781. Retrieved from https://doi.org/10.48550/arXiv.2504.01781

Witkowski, J., Freeman, R., Vaughan, J., Pennock, D., & Krause, A. (2022). Incentive-compatible forecasting competitions. *Management Science*, *69*(3), 1354–1374. Retrieved from https://doi.org/10.1287/mnsc.2022.4410

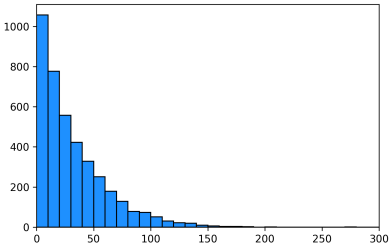
A Alternate uniform distribution



$0.05 < \sigma < 0.35$



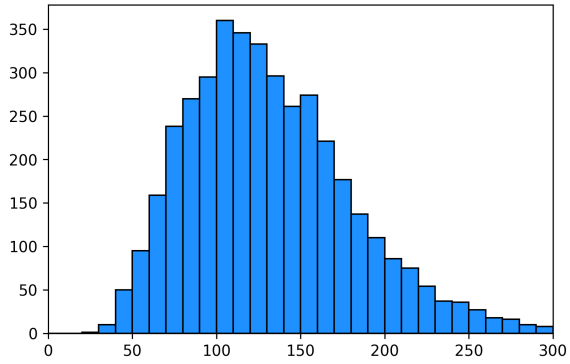
$0.1 < \sigma < 0.4$



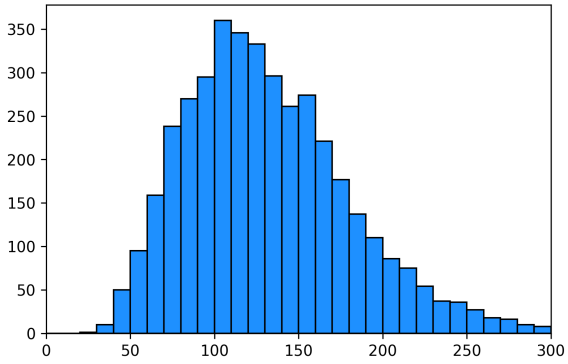
$0.15 < \sigma < 0.45$

Figure A.1: Rank tournament winner, uniform error model, truncation interval to [0, 1]

B Results power score middle scenario



$\beta = 1.1$



$\beta = 1.00001$

Figure B.1: Rank tournament winner Power score with $\beta = 1.1$ and $\beta = 1.00001$, true success probabilities in the middle of [0, 1], for $0 < \sigma < 0.3$

C Code

Binary random variables

Event Class:

```
import numpy.random as npr

class Event:
    def __init__(self, p):
        self.p = p            #the probability that the event happens
        self.outcome = 0      #wether the event happens or not

    def setOutcome(self):
        "samples an outcome for the event"
        self.outcome = npr.binomial(1, self.p)  #outcome of the event

    def Outcome(self):
        "returns the outcome of the event"
        return self.outcome

    def Prob(self):
        "returns the probability of the event happening"
        return self.p
```

Contestant Class:

```
import random
import numpy.random as npr
from math import sqrt, log
import scipy.stats as stats

class Contestant:
    def __init__(self, sigma = 0):
        #contestant has a deviation sigma and a score
        self.sigma = sigma
        self.score = 0

    def calcScore(self, event, sig):
        "calculates the Brier score based on the outcome of an event"
        q = max(0, min(event.Prob() + sig, 1))      #q = p +- sig bounded to the interval [0,1]
        a = q - event.Outcome()
        return a*a                                  #score is (q - outcome event)^2

    #different scoring rules
    def calcScoreLog(self, event, sig):
        "calculates the Logarithmic score based on the outcome of an event"
        q = max(0, min(event.Prob() + sig, 1))      #q = p +- sig bounded to the interval [0,1]
        if event.Outcome() == 1:
            if q == 0:
                return float('inf')
            return -log(q)
        else:
            if q == 1:
                return float('inf')
            return -log(1-q)

    def calcScoreSphere(self, event, sig):
        "calculates the Spherical score based on the outcome of an event"
```

```
        q = max(0, min(event.Prob() + sig, 1))      #q = p +- sig bounded to the interval [0,1]
        norm = sqrt(q*q + (1-q)*(1-q))
        if event.Outcome() == 1:
            return -q / norm
        else:
            return -(1-q) / norm

    def calcScorePseudoSphere(self, event, sig, b):
        "calculates the Pseudospherical score based on the outcome of an event"
        q = max(0, min(event.Prob() + sig, 1))      #q = p +- sig bounded to the interval [0,1]
        norm = (q**b + (1-q)**b)**(1/b)
        if event.Outcome() == 1:
            return -(q / norm)**(b-1)
        else:
            return -((1-q) / norm)**(b-1)

    def calcScorePower(self, event, sig, b):
        "calculates the Power score based on the outcome of an event"
        q = max(0, min(event.Prob() + sig, 1))      #q = p +- sig bounded to the interval [0,1]
        if event.Outcome() == 1:
            return -b*(q**(b-1)) + (b-1)*(q**b + (1-q)**b)
        else:
            return -b*((1-q)**(b-1)) + (b-1)*(q**b + (1-q)**b)


    def calcTotalScoreOriginal(self, events):
        "calculates total score based on a sequence of events in the +- model using the Brier score"
        signs = random.choices([-1,1], k = len(events)) #choose a random sign for sigma
        s = 0                                           #score s start at 0
        for i, event in enumerate(events):              #we run through all events
            s += self.calcScore(event, signs[i]*self.sigma) #and sum the scores of each event
        self.score = s
        return s

    def calcTotalScoreLog(self, events):
        "calculates total score based on a sequence of events in the +- model using the Logarithmic
            ↪ score"
        signs = random.choices([-1,1], k = len(events)) #choose a random sign for sigma
        s = 0                                           #score s start at 0
        for i, event in enumerate(events):              #we run through all events
            s += self.calcScoreLog(event, signs[i]*self.sigma) #and sum the scores of each event
        self.score = s
        return s

    def calcTotalScoreSphere(self, events):
        "calculates total score based on a sequence of events in the +- model using the Spherical score
            ↪ "
        signs = random.choices([-1,1], k = len(events)) #choose a random sign for sigma
        s = 0                                           #score s start at 0
        for i, event in enumerate(events):              #we run through all events
            s += self.calcScoreSphere(event, signs[i]*self.sigma) #and sum the scores of each event
        self.score = s
        return s

    def calcTotalScorePseudoSphere(self, events, b = 5):
        "calculates total score based on a sequence of events in the +- model using the Pseudospherical
            ↪  score"
        signs = random.choices([-1,1], k = len(events)) #choose a random sign for sigma
        s = 0                                           #score s start at 0
        for i, event in enumerate(events):              #we run through all events
```

```
            s += self.calcScorePseudoSphere(event, signs[i]*self.sigma, b) #and sum the scores of each
                ↪ event
        self.score = s
        return s

    def calcTotalScorePower(self, events, b = 50):
        "calculates total score based on a sequence of events in the +- model using the Power score"
        signs = random.choices([-1,1], k = len(events)) #choose a random sign for sigma
        s = 0                                           #score s start at 0
        for i, event in enumerate(events):              #we run through all events
            s += self.calcScorePower(event, signs[i]*self.sigma, b) #and sum the scores of each event
        self.score = s
        return s

    #Different sigma
    def calcTotalScoreExtreme(self, events):
        "calculates total score based on a sequence of events in the +- model where contestants make
            ↪ extreme predictions"
        signs = random.choices([-1,1], k = len(events)) #choose a random sign for sigma
        s = 0                                           #score s start at 0
        for i, event in enumerate(events):              #we run through all events
            s += self.calcScoreExtreme(event, signs[i]*self.sigma) #and sum the scores of each event
        self.score = s
        return s

    def calcScoreExtreme(self, event, sig):
        "calculates the Brier score based on the outcome of an event when contestants make extreme
            ↪ predictions"
        if event.Prob() + sig < 0.5:
            q = 0
        else:
            q = 1
        a = q - event.Outcome()
        return a*a                                  #score is (q - outcome event)^2

    def calcTotalScoreUniform(self, events):
        "calculates total score based on a sequence of events in the uniform model"
        a = sqrt(3)*self.sigma
        sigs = npr.uniform(-a, a, len(events)) #choose a random sign for sigma
        s = 0                                                   #score s start at 0
        for i, event in enumerate(events):                      #we run through all events
            s += self.calcScore(event, sigs[i])                 #and sum the scores of each event
        self.score = s
        return s

    def calcTotalScoreUniform2(self, events):
        "calculates total score based on a sequence of events in the alternate uniform model"
        s = 0                                           #score s start at 0
        for i, event in enumerate(events):              #we run through all events
            s += self.calcScoreUniform2(event)          #and sum the scores of each event
        self.score = s
        return s

    def calcScoreUniform2(self, event):
        "calculates a score based on an event in the alternate uniform model"
        sig = sqrt(3)*self.sigma
        a = max(event.Prob() - sig, 0)
        b = min(event.Prob() + sig, 1)
        q = npr.uniform(a, b)
        z = q - event.Outcome()
```

```
        return z*z                                          #score is (q - outcome event)^2

    def calcTotalScoreBeta(self, events):
        "calculates total score based on a sequence of events in the varied Beta model"
        if self.sigma != 0:
            a = 1/8 * 1/(self.sigma**2) - 1/2
            b = a
            sigs = stats.beta.rvs(a, b, loc = -0.5, size = len(events)) #choose a random sign for sigma
        else:
            sigs = [0 for i in range(len(events))]
        s = 0                                               #score s start at 0
        for i, event in enumerate(events):                  #we run through all events
            s += self.calcScore(event, sigs[i])             #and sum the scores of each event
        self.score = s
        return s

    def calcTotalScoreBetaFixed(self, events, alpha, beta):
        "calculates total score based on a sequence of events in the rescaled Beta model"
        if self.sigma != 0:
            std = stats.beta.std(alpha, beta)
            mean = stats.beta.mean(alpha, beta)
            sigs = stats.beta.rvs(alpha, beta, loc = -mean*1/std*self.sigma, scale = 1/std*self.sigma,
                ↪ size = len(events)) #choose a random sign for sigma
        else:
            sigs = [0 for i in range(len(events))]
        s = 0                                               #score s start at 0
        for i, event in enumerate(events):                  #we run through all events
            s += self.calcScore(event, sigs[i])             #and sum the scores of each event
        self.score = s
        return s

    def Score(self):
        "returns the score of a contestant"
        return self.score

    def Sigma(self):
        "returns the deviation (sigma) of a contestant"
        return self.sigma
```

Tournament Class:

```
mport numpy as np
from math import floor
from Event import Event
from Contestant import Contestant
from scipy.stats import rankdata
import random

class Tournament:
    def __init__(self, probabilities, contestants):
        self.n = len(probabilities)                         #number of events
        self.m = len(contestants)                           #number of contestants
        self.probabilities = probabilities                  #list of probabilities for each event
        self.events = [Event(p) for p in self.probabilities] #create list of events
        self.contestants = contestants                      #list of contestants

    def setStandard(self):
        "set standard tournament with 100 events (p = 0.05, 0.15, ..., 0.95) (each prob occurs 10x) \
            300 contestants with sigma evenly distributed on [0.05, 0.3]"
```

```
        self.n = 100
        self.m = 300
        self.probabilities = [round(0.05 + 0.1*floor(i/10), 2) for i in range(0,self.n)]
        self.events = [Event(p) for p in self.probabilities]
        self.contestants = [Contestant(round(0.05 + 0.3*i/self.m, 3)) for i in range(self.m)]

    def setStandardSigma(self, low, high, m = 300):
        "specifiy the tournament to have m contestants with sigma evenly distributed on [low, high]"
        self.m = m
        self.contestants = [Contestant(round(low + (high-low)*i/self.m, 3)) for i in range(self.m)]

    def setStandardProb(self):
        "specifiy the tournament to have 100 events (p = 0.05, 0.15, ..., 0.95) (each prob occurs 10x)"
        self.n = 100
        self.probabilities = [round(0.05 + 0.1*floor(i/10), 2) for i in range(0,self.n)]
        self.events = [Event(p) for p in self.probabilities]

    def run(self, model = 'original', q = 0.5, b = 5, alpha = 1, beta = 1):
        "runs the tournament"
        #set the outcomes for all events
        for e in self.events:
            e.setOutcome()

        #uses the appropriate model for contestant predictions and scores
        if model == 'original':
            for c in self.contestants:
                c.calcTotalScoreOriginal(self.events)
        if model == 'uniform':
            for c in self.contestants:
                c.calcTotalScoreUniform(self.events)
        if model == 'uniform2':
            for c in self.contestants:
                c.calcTotalScoreUniform2(self.events)
        if model == 'beta':
            for c in self.contestants:
                c.calcTotalScoreBeta(self.events)
        if model == 'betaFixed':
            for c in self.contestants:
                c.calcTotalScoreBetaFixed(self.events, alpha, beta)
        if model == 'extreme':
            for c in self.contestants:
                c.calcTotalScoreExtreme(self.events)
        if model == 'extremeComparison':
            for index, c in enumerate(self.contestants):
                if index % 2 == 1:
                    c.calcTotalScoreExtreme(self.events)
                else:
                    c.calcTotalScoreOriginal(self.events)
        if model == 'log':
            for c in self.contestants:
                c.calcTotalScoreLog(self.events)
        if model == 'sphere':
            for c in self.contestants:
                c.calcTotalScoreSphere(self.events)
        if model == 'pseudosphere':
            for c in self.contestants:
                c.calcTotalScorePseudoSphere(self.events, b)
        if model == 'power':
            for c in self.contestants:
                c.calcTotalScorePower(self.events, b)
```

```
79
80      def probabilities(self):
81          "returns the success probabilities of the tournament"
82          return [event.Prob() for event in self.events]
83
84      def outcomes(self):
85          "returns the outcomes of the events in the tournament"
86          return [event.Outcome() for event in self.events]
87
88      def scoreContestants(self):
89          "returns the scores of the contestants in the tournament"
90          return [c.Score() for c in self.contestants]
91
92      def sigmaContestants(self):
93          "returns the deviations(sigma) of the contestants in the tournament"
94          return [c.Sigma() for c in self.contestants]
95
96      def rankWinner(self):
97          "returns the rank of the contestant with the lowest score (the winner of the tournament)"
98          #return np.flatnonzero(self.scoreContestants() == np.min(self.scoreContestants()))
99          return np.argmin(self.scoreContestants())
100
101     def allWinners(self):
102         "returns all contestants with the same lowest score)"
103         return np.flatnonzero(self.scoreContestants() == np.min(self.scoreContestants()))
104
105     def randomWinner(self):
106         "if multiple people have the same lowest score, we randomly select a winner from those"
107         return random.choice(self.allWinners())
108
109     def rankFinishPos(self, n):
110         "returns the rank of the contestant with the lowest score (the winner of the tournament)"
111         return np.argsort(self.scoreContestants())[n]
112
113     def rankTopN(self, n=10):
114         "returns the rank of top n finishers"
115         return np.argsort(self.scoreContestants())[0:n]
116
117     def rankAll(self):
118         "returns the rank for all finish positions"
119         return np.argsort(self.scoreContestants())
120
121     def finishPosAll(self):
122         "returns the finish position for all contestants"
123         return (rankdata(self.scoreContestants()) - 1).astype(int)
```

Simulate File:

```
1  import matplotlib.pyplot as plt
2  import time
3  from Tournament import Tournament
4  import numpy.random as npr
5  import numpy as np
6  import scipy.stats as stats
7  from datetime import datetime
8
9  def Simulate(n = 4000, low = 0, high = 0.3, m = 300, topN = False, N = 10, model = 'original', prob =
       ↪ 'standard', q = 0.5, b = 5, filename = None, alpha = 1, beta = 1):
10     ranks = []
```

```
11
12      p = []
13
14      #save results in a text file
15      if filename != None:
16          f = open(f'allRanks{filename}.txt', 'w')
17          f1 = open(f'Scores{filename}.txt', 'w')
18          f2 = open(f'finishPos{filename}.txt', 'w')
19
20      #options for the true success probilities
21      if prob == 'double':
22          p = 100*[0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95]
23      if prob == 'left':
24          p = 20*[0.05, 0.10, 0.15, 0.20, 0.25]
25      if prob == 'middle':
26          p = 20*[0.4, 0.45, 0.5, 0.55, 0.6]
27      if prob == 'right':
28          p = 20*[0.75, 0.80, 0.85, 0.90, 0.95]
29      if prob == 'leftright':
30          p = 10*[0.05, 0.10, 0.15, 0.20, 0.25, 0.75, 0.80, 0.85, 0.90, 0.95]
31      if prob == 'precise':
32          p = [i/100 for i in range(100)]
33      if prob == 'uniform':
34          alpha, beta = 0, 1
35          p = npr.uniform(alpha, beta,100)
36
37          plt.figure(1)
38          plt.hist(p, bins = 15, color = 'c', edgecolor='k', density = True)
39
40          points = np.linspace(alpha, beta,100)
41          pdf = stats.uniform.pdf(points, alpha, beta)
42          plt.plot(points, pdf, color='r')
43      if prob == 'beta':
44          alpha, beta = 1, 1
45          p = npr.beta(alpha, beta,100)
46
47          plt.figure(1)
48          plt.hist(p, bins = 15, color = 'c', edgecolor='k', density = True)
49
50          points = np.linspace(0, 1,100)
51          pdf = stats.beta.pdf(points, alpha, beta)
52          plt.plot(points, pdf, color='r')
53
54      t = Tournament(p, [])
55      t.setStandardSigma(low, high, m)
56
57      if prob == 'standard':
58          t.setStandardProb()
59
60      if model == 'extremeComparison':
61          even = []
62          odd = []
63
64      if topN == False:
65          for i in range(n):
66              t.run(model, q, b)
67              winner = t.randomWinner()
68              ranks.append(winner)
69              #ranks.append(t.rankFinishPos(14))
70              if model == 'extremeComparison':
```

```
                if winner % 2 == 1:
                    odd.append(winner)
                else:
                    even.append(winner)
            if filename != None:
                f.write(str(list(t.finishPosAll())) + '\n')
                f1.write(str(t.scoreContestants()) + '\n')
                a = str(list(t.rankAll()))
                l1 = a.replace('[', '')
                l2 = l1.replace(']', '')
                items = l2.split(',')
                a = [float(item) for item in items]
                if len(a) != 300:
                    print(len(a))
                f2.write(str(list(t.rankAll())) + '\n')
    if topN == True:
        for i in range(n):
            t.run(model, q, b)
            ranks.extend(t.rankTopN(N))
    plt.figure(2, dpi = 300)
    plt.hist(ranks, bins = range(0, 300 + 10, 10), color = 'dodgerblue', edgecolor='k')
    plt.xlim(xmin=0, xmax = 300)

    if model == 'extremeComparison':
        plt.figure(3, dpi = 300)
        plt.hist([even, odd], bins = range(0, 300 + 10, 10), color = ['dodgerblue', 'orange'], label =
            ↪ ['Normal predictions', 'Extreme predictions'])
        plt.xlim(xmin=0, xmax = 300)
        plt.legend(loc = 'upper right')



    plt.show()

    if filename != None:
        f.close()
        f1.close()
        f2.close()

print(datetime.now())
Simulate(4000)

#variations: original
#sigma distribution: uniform, uniform2, normal, beta, betaFixed, extreme, extremeComparison
#score: log, sphere, pseudosphere, power
```

Multi-categorical random variables

Event Class:

```
import numpy as np

class EventMC:
    def __init__(self, p):
        self.p = p            #the probability that the event happens
        self.n = len(self.p)
        self.outcome = 0      #outcome event
        if round(sum(self.p), 5) != 1:
            print('probabilities dont sum to 1', self.p, 'sum:', sum(len(self.p)))
```

```
        for prob in self.p:
            if prob < 0 or prob > 1:
                print('probs not in range [0, 1]', self.p)


    def setOutcome(self):
        "samples an outcome for the event"
        self.outcome = np.random.choice(self.n, 1, p = self.p)

    def Outcome(self):
        "returns the outcome of the event"
        return self.outcome

    def Prob(self):
        "returns the probability of the event happening"
        return self.p
```

Contestant Class:

```
import random
from math import sqrt, log
import scipy.special as scip

class ContestantMC:
    def __init__(self, sigma = 0):
        #contestant has a deviation sigma and a score
        self.sigma = sigma
        self.score = 0

    def calcProbs(self, event, sig):
        totalProb = 0
        probs = []
        for index, prob in enumerate(sig):
            q = max(0, min(event.Prob()[index] + sig[index], 1 - totalProb))       #q = p +- sig
                ↪ bounded to the interval [0,1]
            totalProb += q
            probs.append(q)

        probs.append(1 - totalProb)

        if sum(probs) != 1:
            print(probs)

        return probs

    def calcScoreBrier(self, event, sig):
        probs = self.calcProbs(event, sig)

        outcome = int(event.Outcome())

        score = 0
        for index, prob in enumerate(probs):
            y = float(outcome == index)
            score += (prob - y)**2

        return score

    def calcScoreCRPS(self, event, sig):
        probs = self.calcProbs(event, sig)
```

```
        category = len(probs) - 1

        outcome = int(event.Outcome())

        score = 0

        for i in range(outcome):
            score += probs[i]**2

        for i in range(outcome, category):
            score += (probs[i] - 1)**2

        return score

    def calcScoreLog(self, event, sig):
        probs = self.calcProbs(event, sig)

        outcome = int(event.Outcome())

        if probs[outcome] == 0:
            return float('inf')
        return -log(probs[outcome])

    def calcScoreSphere(self, event, sig):
        probs = self.calcProbs(event, sig)

        norm = 0
        for p in probs:
            norm += p*p
        norm = sqrt(norm)

        outcome = int(event.Outcome())

        return -probs[outcome]/norm


    def calcScorePseudoSphere(self, event, sig, beta):
        probs = self.calcProbs(event, sig)

        norm = 0
        for p in probs:
            norm += p**(beta)
        norm = norm**(1/beta)

        outcome = int(event.Outcome())

        return -(probs[outcome]/norm)**(beta - 1)

    def calcScorePower(self, event, sig, beta):
        probs = self.calcProbs(event, sig)

        norm = 0
        for p in probs:
            norm += p**(beta)

        outcome = int(event.Outcome())

        return -beta*(probs[outcome])**(beta - 1) + (beta - 1)*norm

    def calcTotalScore(self, events, score = 'Brier', distance = 'TVD', beta = 1.01):
```

```
        category = events[0].n
        signs = random.choices([-1,1], k = len(events)*(category-1)) #choose a random sign for sigma
        s = 0                                                        #score s start at 0

        factor = 1
        if distance == 'TVD':
            factor = 2/self.compFactorTVD(category - 1)
        if distance == 'Wasserstein':
            factor = 1/self.compFactorWasserStein(category - 1)
        #print(factor)

        if score == 'Brier':
            for i, event in enumerate(events):              #we run through all events
                lst = [signs[i+j*(category-1)]*self.sigma*factor for j in range(category - 1)]
                s += self.calcScoreBrier(event, lst) #and sum the scores of each event
        if score == 'CRPS':
            for i, event in enumerate(events):              #we run through all events
                lst = [signs[i+j*(category-1)]*self.sigma*factor for j in range(category - 1)]
                s += self.calcScoreCRPS(event, lst) #and sum the scores of each event
        if score == 'Log':
            for i, event in enumerate(events):              #we run through all events
                lst = [signs[i+j*(category-1)]*self.sigma*factor for j in range(category - 1)]
                s += self.calcScoreLog(event, lst) #and sum the scores of each event
        if score == 'Sphere':
            for i, event in enumerate(events):              #we run through all events
                lst = [signs[i+j*(category-1)]*self.sigma*factor for j in range(category - 1)]
                s += self.calcScoreSphere(event, lst) #and sum the scores of each event
        if score == 'Power':
            for i, event in enumerate(events):              #we run through all events
                lst = [signs[i+j*(category-1)]*self.sigma*factor for j in range(category - 1)]
                s += self.calcScorePower(event, lst, beta) #and sum the scores of each event
        if score == 'PseudoSphere':
            for i, event in enumerate(events):              #we run through all events
                lst = [signs[i+j*(category-1)]*self.sigma*factor for j in range(category - 1)]
                s += self.calcScorePseudoSphere(event, lst, beta) #and sum the scores of each event

        self.score = s
        return s

    def compFactorWasserStein(self, C):
        s = 0
        for i in range(C):
            for j in range(i + 2):
                s += (scip.binom(i + 1, j)*abs(2*j - (i+1)))/(2**(i+1))
        return s

    def compFactorTVD(self, C):
        s = 0
        for i in range(C+1):
            s += (scip.binom(C, i)*abs(2*i - C)/(2**C))
        return s + C


    def Score(self):
        "returns the score of a contestant"
        return self.score

    def Sigma(self):
        "returns the deviation (sigma) of a contestant"
        return self.sigma
```

Tournament Class:

```
import numpy as np
from math import floor
from EventMultiCategory import EventMC
from ContestantMultiCategory import ContestantMC
from scipy.stats import rankdata

class TournamentMC:
    def __init__(self, probabilities, contestants):
        self.n = len(probabilities)                         #number of events
        self.m = len(contestants)                           #number of contestants
        self.probabilities = probabilities                  #list of probabilities for each event
        self.events = [EventMC(p) for p in self.probabilities] #create list of events
        self.contestants = contestants                      #list of contestants

    def setStandard(self):
        "set standard tournament with 100 events (p = 0.05, 0.15, ..., 0.95) (each prob occurs 10x) \
            300 contestants with sigma evenly distributed on [0.05, 0.3]"
        self.n = 100
        self.m = 300
        self.probabilities = [round(0.05 + 0.1*floor(i/10), 2) for i in range(0,self.n)]
        self.events = [EventMC(p) for p in self.probabilities]
        self.contestants = [ContestantMC(round(0.05 + 0.3*i/self.m, 3)) for i in range(self.m)]

    def setStandardSigma(self, low, high, m = 300):
        "set standard tournament with 100 events (p = 0.05, 0.15, ..., 0.95) (each prob occurs 10x) \
            m contestants with sigma evenly distributed on [low, high]"
        self.m = m
        self.contestants = [ContestantMC(round(low + (high-low)*i/self.m, 3)) for i in range(self.m)]

    def setStandardProb(self, category):
        "set standard tournament with 100 events (p = 0.05, 0.15, ..., 0.95) (each prob occurs 10x) \
            m contestants with sigma evenly distributed on [low, high]"

        if category == 2:
            probs = [round(0.05 + 0.1*floor(i/10), 2) for i in range(0, 100)]
            self.probabilities = [[round(p, 2), round(1-p, 2)] for p in probs]

        if category == 3:
            self.probabilities = 2*[[0.05, 0.05, 0.9],
                                    [0.05, 0.15, 0.8],
                                    [0.05, 0.25, 0.7],
                                    [0.05, 0.35, 0.6],
                                    [0.05, 0.45, 0.5],
                                    [0.05, 0.55, 0.4],
                                    [0.05, 0.65, 0.3],
                                    [0.05, 0.75, 0.2],
                                    [0.05, 0.85, 0.1],
                                    [0.15, 0.15, 0.7],
                                    [0.15, 0.25, 0.6],
                                    [0.15, 0.35, 0.5],
                                    [0.15, 0.45, 0.4],
                                    [0.15, 0.55, 0.3],
                                    [0.15, 0.65, 0.2],
                                    [0.15, 0.75, 0.1],
                                    [0.25, 0.25, 0.5],
                                    [0.25, 0.35, 0.4],
                                    [0.25, 0.45, 0.3],
                                    [0.25, 0.55, 0.2],
```

```
                            [0.25, 0.65, 0.1],
                            [0.35, 0.35, 0.3],
                            [0.35, 0.45, 0.2],
                            [0.35, 0.55, 0.1],
                            [0.45, 0.45, 0.1]] + 2*[[0.05, 0.05, 0.9],
                                                    [0.15, 0.05, 0.8],
                                                    [0.25, 0.05, 0.7],
                                                    [0.35, 0.05, 0.6],
                                                    [0.45, 0.05, 0.5],
                                                    [0.55, 0.05, 0.4],
                                                    [0.65, 0.05, 0.3],
                                                    [0.75, 0.05, 0.2],
                                                    [0.85, 0.05, 0.1],
                                                    [0.15, 0.15, 0.7],
                                                    [0.25, 0.15, 0.6],
                                                    [0.35, 0.15, 0.5],
                                                    [0.45, 0.15, 0.4],
                                                    [0.55, 0.15, 0.3],
                                                    [0.65, 0.15, 0.2],
                                                    [0.75, 0.15, 0.1],
                                                    [0.25, 0.25, 0.5],
                                                    [0.35, 0.25, 0.4],
                                                    [0.45, 0.25, 0.3],
                                                    [0.55, 0.25, 0.2],
                                                    [0.65, 0.25, 0.1],
                                                    [0.35, 0.35, 0.3],
                                                    [0.45, 0.35, 0.2],
                                                    [0.55, 0.35, 0.1],
                                                    [0.45, 0.45, 0.1]]
        self.n = len(self.probabilities)
        self.events = [EventMC(p) for p in self.probabilities]

    def run(self, score = 'Brier', distance = 'TVD', beta = 1.01):
        "runs the tournament"

        for e in self.events:
            e.setOutcome()

        for c in self.contestants:
            c.calcTotalScore(self.events, score, distance, beta)

    def probabilities(self):
        return [event.Prob() for event in self.events]

    def outcomes(self):
        "returns the outcomes of the events in the tournament"
        return [event.Outcome() for event in self.events]

    def scoreContestants(self):
        "returns the scores of the contestants in the tournament"
        return [c.Score() for c in self.contestants]

    def sigmaContestants(self):
        "returns the deviations(sigma) of the contestants in the tournament"
        return [c.Sigma() for c in self.contestants]

    def rankWinner(self):
        #return np.flatnonzero(self.scoreContestants() == np.min(self.scoreContestants()))
        return np.argmin(self.scoreContestants())

```

```
    def rankFinishPos(self, n):
        return np.argsort(self.scoreContestants())[n]

    def rankTopN(self, n=10):
        return np.argsort(self.scoreContestants())[0:n]

    def rankAll(self):
        return np.argsort(self.scoreContestants())

    def finishPosAll(self):
        return (rankdata(self.scoreContestants()) - 1).astype(int)
```

Simulate File:

```
import matplotlib.pyplot as plt
import time
from TournamentMultiCategory import TournamentMC
import numpy.random as npr
from datetime import datetime

def defineProb(category, sort = 'uniform'):
    p = []
    if sort == 'uniformSkewed':
        sample = npr.uniform(0, 1, category*100)
        for i in range(100):
            lst = [sample[i+j*category] for j in range(category)]
            s = sum(lst)
            for j in range(category):
                lst[j] = lst[j]/s
            p.append(lst)

    if sort == 'uniform':
        sample = npr.uniform(0, 1, (category-1)*100)
        for i in range(100):
            lst = [sample[i+j*category] for j in range(category - 1)]
            lst.append(0)
            lst = sorted(lst)
            lst.append(1)

            new_lst = []
            for i in range(1, category + 1):
                new_lst.append(lst[i] - lst[i - 1])

            if sum(new_lst) != 1:
                print(new_lst)

            p.append(new_lst)
    return p


def Simulate(n = 4000, n2 = 400, low = 0, high = 0.3, m = 300, category = 3, topN = False, N = 10,
    ↪ score = 'Brier', distance ='TVD', prob = 'standard', beta = 1.01, filename = None):
    ranks = []

    if filename != None:
        f = open(f'allRanks{filename}.txt', 'w')
        f1 = open(f'Scores{filename}.txt', 'w')
        f2 = open(f'finishPos{filename}.txt', 'w')

```

```
    for i in range(n):
        if i % (n // n2) == 0:
            p = defineProb(category, prob)

            t = TournamentMC(p, [])
            t.setStandardSigma(low, high, m)
            if prob == 'standard':
                t.setStandardProb(category)
        t.run(score, distance, beta)
        ranks.append(t.rankWinner())
        if filename != None:
            f.write(str(list(t.finishPosAll())) + '\n')
            f1.write(str(t.scoreContestants()) + '\n')
            f2.write(str(list(t.rankAll())) + '\n')
    plt.figure(2, dpi = 600)
    plt.hist(ranks, bins = range(0, 300 + 10, 10), color = 'dodgerblue', edgecolor='k')
    plt.xlim(xmin=0, xmax = 300)
    plt.show()

    if filename != None:
        f.close()
        f1.close()
        f2.close()


print(datetime.now())
Simulate(4000)

#variations:
#score: Brier, CRPS, Log, Sphere, Power, PseudoSphere,
#distance: TVD, Wasserstein
```

Continuous random variables

Event Class:

```
import scipy.stats as stats
from math import sqrt

class EventD:
    def __init__(self, param, tpe):
        self.param = param          #list of parameters
        self.tpe = tpe              #event type
        self.outcome = 0            #outcome event

    def setOutcome(self):
        "samples an outcome for the event"
        if self.tpe == 'normal':
            self.outcome = stats.norm.rvs(loc = self.param[0], scale = sqrt(self.param[1]), size = 1)
        if self.tpe == 'uniform':
            a = self.param[0]
            self.outcome = 2*a*stats.uniform.rvs(size = 1) - a

    def Outcome(self):
        "returns the outcome of the event"
        return self.outcome

    def Prob(self):
        "returns the probability of the event happening"
```

```
        return self.param

    def Tpe(self):
        "returns the probability of the event happening"
        return self.tpe
```

Contestant Class:

```
import random
from math import sqrt, log, pi
from scipy import stats

class ContestantD:
    def __init__(self, sigma = 0):
        #contestant has a deviation sigma and a score
        self.sigma = sigma
        self.score = 0

    def ComputeIntegral(self, tpe, param):
        integral = 0
        if tpe == 'uniform':
            integral = 1/(2*param[0])

        if tpe == 'normal':
            integral = 1/(2*sqrt(param[1]*pi))

        return integral

    def ComputeIntegralB(self, tpe, param, b = 1.01):
        integral = 0
        if tpe == 'normal':
            integral = (1/(sqrt(2*param[1]*pi)))**(b-1) * 1/sqrt(b)

        return integral

    def ComputeParam(self, event, sig):
        param = []
        if event.tpe == 'uniform':
            a = event.param[0]
            if sig == 1:
                param = (a / (1 - self.sigma),)
            if sig == -1:
                param = (a * (1 - self.sigma),)

        if event.tpe == 'normal':
            sigmasquare = event.param[1]
            mu = 2*sqrt(sigmasquare)*stats.norm.ppf((self.sigma + 1)/2, loc=0, scale=1)

            if sig == 1:
                param = (mu, sigmasquare)
            if sig == -1:
                param = (-mu, sigmasquare)

        return param

    def calcScoreBrier(self, event, sig):
        "calculates a score based on an event"

        param = self.ComputeParam(event, sig)
```

```
        integral = self.ComputeIntegral(event.tpe, param)

        tpe = event.tpe

        outcome = float(event.Outcome())

        if tpe == 'uniform':
            py = stats.uniform.pdf(outcome, loc = -param[0], scale = 2*param[0])
        if tpe == 'normal':
            py = stats.norm.pdf(outcome, loc = param[0], scale = sqrt(param[1]))

        #print('outcome:', outcome, 'py:', py, 'param:', param)
        return -2*py + integral

    def calcScoreSphere(self, event, sig):
        "calculates a score based on an event"
        param = self.ComputeParam(event, sig)
        integral = self.ComputeIntegral(event.tpe, param)

        tpe = event.tpe

        outcome = float(event.Outcome())

        if tpe == 'uniform':
            py = stats.uniform.pdf(outcome, loc = -param[0], scale = 2*param[0])
        if tpe == 'normal':
            py = stats.norm.pdf(outcome, loc = param[0], scale = sqrt(param[1]))

        return -py/sqrt(integral)

    def calcScoreCRPS(self, event, sig):
        "calculates a score based on an event"
        param = self.ComputeParam(event, sig)

        tpe = event.tpe

        outcome = float(event.Outcome())

        if tpe == 'uniform':
            a = param[0]
            if outcome < -a or outcome > a:
                return 2*a/3
            else:
                return -outcome**3/(12*a**2) + (a + outcome)**3/(12*a**2) + outcome**2/(4*a) + a/12 -
                    ↪ outcome/4

        if tpe == 'normal':
            mu = param[0]
            sd = sqrt(param[1])

            c = (outcome - mu)/sd

            return sd*(c * (2*stats.norm.cdf(c) - 1) + 2*stats.norm.pdf(c) - 1/sqrt(pi))

    def calcScoreLog(self, event, sig):
        "calculates a score based on an event"
        param = self.ComputeParam(event, sig)
        #print('param:', param, 'sigma:', self.sigma, 'event:', event.param)

        tpe = event.tpe
```

```
111
112         outcome = float(event.Outcome())
113
114         if tpe == 'uniform':
115             py = stats.uniform.pdf(outcome, loc = -param[0], scale = 2*param[0])
116         if tpe == 'normal':
117             py = stats.norm.pdf(outcome, loc = param[0], scale = sqrt(param[1]))
118
119         if py == 0:
120             return float('inf')
121         return -log(py)
122
123     def calcScorePower(self, event, sig, beta):
124         "calculates a score based on an event"
125
126         param = self.ComputeParam(event, sig)
127         integral = self.ComputeIntegralB(event.tpe, param, beta)
128
129         tpe = event.tpe
130
131         outcome = float(event.Outcome())
132
133         if tpe == 'normal':
134             py = stats.norm.pdf(outcome, loc = param[0], scale = sqrt(param[1]))
135
136         #print('outcome:', outcome, 'py:', py, 'param:', param)
137         return -beta*py**(beta-1) + (beta-1)*integral
138
139     def calcScorePseudosphere(self, event, sig, beta):
140         "calculates a score based on an event"
141
142         param = self.ComputeParam(event, sig)
143         integral = self.ComputeIntegralB(event.tpe, param, beta)
144
145         norm = integral**(1/beta)
146
147         tpe = event.tpe
148
149         outcome = float(event.Outcome())
150
151         if tpe == 'normal':
152             py = stats.norm.pdf(outcome, loc = param[0], scale = sqrt(param[1]))
153
154         #print('outcome:', outcome, 'py:', py, 'param:', param)
155         return -(py/norm)**(beta - 1)
156
157     def calcTotalScore(self, events, score = 'Brier', beta = 1.01):
158         "calculates total score based on a sequence of events in the +- model "
159         signs = random.choices([-1,1], k = len(events)) #choose a random sign for sigma
160         s = 0                                           #score s start at 0
161
162         if score == 'Brier':
163             for i, event in enumerate(events):              #we run through all events
164                 s += self.calcScoreBrier(event, signs[i]) #and sum the scores of each event
165         if score == 'CRPS':
166             for i, event in enumerate(events):              #we run through all events
167                 s += self.calcScoreCRPS(event, signs[i]) #and sum the scores of each event
168         if score == 'Log':
169             for i, event in enumerate(events):              #we run through all events
170                 s += self.calcScoreLog(event, signs[i]) #and sum the scores of each event
```

```
        if score == 'Sphere':
            for i, event in enumerate(events):              #we run through all events
                s += self.calcScoreSphere(event, signs[i]) #and sum the scores of each event
        if score == 'Pseudosphere':
            for i, event in enumerate(events):              #we run through all events
                s += self.calcScorePseudosphere(event, signs[i], beta) #and sum the scores of each event
        if score == 'Power':
            for i, event in enumerate(events):              #we run through all events
                s += self.calcScorePower(event, signs[i], beta) #and sum the scores of each event


        self.score = s
        return s


    def Score(self):
        "returns the score of a contestant"
        return self.score

    def Sigma(self):
        "returns the deviation (sigma) of a contestant"
        return self.sigma
```

Tournament Class:

```
import numpy as np
from math import floor
from EventDensity import EventD
from ContestantDensity import ContestantD
from scipy.stats import rankdata

class TournamentD:
    def __init__(self, probabilities, contestants):
        self.n = len(probabilities)                        #number of events
        self.m = len(contestants)                          #number of contestants
        self.events = [EventD(p) for p in probabilities] #create list of events
        self.contestants = contestants                     #list of contestants

    def setStandard(self):
        "set standard tournament with 100 events (p = 0.05, 0.15, ..., 0.95) (each prob occurs 10x) \
            300 contestants with sigma evenly distributed on [0.05, 0.3]"
        self.n = 100
        self.m = 300
        self.probabilities = [round(0.05 + 0.1*floor(i/10), 2) for i in range(0,self.n)]
        self.events = [EventD(p) for p in self.probabilities]
        self.contestants = [ContestantD(round(0.05 + 0.3*i/self.m, 3)) for i in range(self.m)]

    def setStandardSigma(self, low, high, m = 300):
        "set standard tournament with 100 events (p = 0.05, 0.15, ..., 0.95) (each prob occurs 10x) \
            m contestants with sigma evenly distributed on [low, high]"
        self.m = m
        self.contestants = [ContestantD(round(low + (high-low)*i/self.m, 3)) for i in range(self.m)]

    def setStandardProb(self, tpe, N = 10):
        "set standard tournament with 100 events (p = 0.05, 0.15, ..., 0.95) (each prob occurs 10x) \
            m contestants with sigma evenly distributed on [low, high]"

        self.events = []
        if tpe == 'uniform':
```

```
            for i in range(100):
                if N == 'no':
                    param = (1,)
                elif N == 'yes':
                    param = (i+1,)
                else:
                    param = (0.5 + i/N,)
                self.events.append(EventD(param, tpe))

        if tpe == 'normal':
            for i in range(100):
                #param = (0, 1)
                if N == 'no':
                    param = (0, 1)
                else:
                    param = (0, 0.5 + i/N)
                self.events.append(EventD(param, tpe))

        self.n = len(self.events)

    def run(self, score = 'Brier', beta = 1.01):
        "runs the tournament"

        for e in self.events:
            e.setOutcome()

        for c in self.contestants:
            c.calcTotalScore(self.events, score, beta)

    def probabilities(self):
        return [event.Prob() for event in self.events]

    def outcomes(self):
        "returns the outcomes of the events in the tournament"
        return [event.Outcome() for event in self.events]

    def scoreContestants(self):
        "returns the scores of the contestants in the tournament"
        return [c.Score() for c in self.contestants]

    def sigmaContestants(self):
        "returns the deviations(sigma) of the contestants in the tournament"
        return [c.Sigma() for c in self.contestants]

    def rankWinner(self):
        "returns the rank of the contestant with the lowest score (the winner of the tournament)"
        return np.argmin(self.scoreContestants())

    def rankFinishPos(self, n):
        return np.argsort(self.scoreContestants())[n]

    def rankTopN(self, n=10):
        return np.argsort(self.scoreContestants())[0:n]

    def rankAll(self):
        return np.argsort(self.scoreContestants())

    def finishPosAll(self):
        return (rankdata(self.scoreContestants()) - 1).astype(int)
```

Simulate File:

```
import matplotlib.pyplot as plt
import time
from TournamentDensity import TournamentD
from datetime import datetime

def Simulate(n = 4000, low = 0, high = 0.3, m = 300, tpe = 'normal', N = 10, score = 'Brier', prob = '
    ↪ standard', beta = 1.01, filename = None):
    ranks = []
    p = []

    if filename != None:
        f = open(f'allRanks{filename}.txt', 'w')
        f1 = open(f'Scores{filename}.txt', 'w')
        f2 = open(f'finishPos{filename}.txt', 'w')

    t = TournamentD(p, [])
    t.setStandardSigma(low, high, m)

    if prob == 'standard':
        t.setStandardProb(tpe, N)

    for i in range(n):
        t.run(score, beta)
        ranks.append(t.rankWinner())
        #ranks.append(t.rankFinishPos(14))
        if filename != None:
            f.write(str(list(t.finishPosAll())) + '\n')
            f1.write(str(t.scoreContestants()) + '\n')
            f2.write(str(list(t.rankAll())) + '\n')
    plt.figure(2, dpi = 300)
    plt.hist(ranks, bins = range(0, 300 + 10, 10), color = 'dodgerblue', edgecolor='k')
    plt.xlim(xmin=0, xmax = 300)
    plt.show()

    if filename != None:
        f.close()
        f1.close()
        f2.close()


print(datetime.now())
#Simulate(4000)
```