Harnessing Meta and Weighted Ensemble Learners to Enhance Horse Race Forecasting

Li Zhang
1009010026
lijian.zhang

Mark Zhang
1009007959
markz.zhang

1 Introduction

Data analysis is crucial in sports for decision-making and predicting outcomes. While popular in many sports, horse racing remains underexplored. Millions enjoy horse racing, which blends fast-paced action with strategic betting. Despite the simple goal of picking a winner, numerous variables make prediction challenging.

Our focus is on races at Woodbine Racetrack in Toronto to standardize variables and provide a consistent baseline for our models. Using 2023 race results and past performance data from Equibase, we aim to develop predictive algorithms for horse racing.

2 Motivation

Current approaches in this domain are predominately focused on traditional statistical models. Often times these statistical approaches assume some type of independence or underlying distribution. Instead of making these heavy assumptions, we aim to create a learning process that balances its assumptions to capture all the complex interaction in races.

3 Previous Literature

3.1 Statistical Models

Statistical models for horse racing often rely on simplifying assumptions. Ali (1998) [1] used normal (Thurstonian) and gamma probability models to estimate win probabilities. Parameters were estimated using maximum likelihood estimation (MLE) on historical data. Each horse's empirical win probability $p_n(i)$ was derived from race odds. The performance score z_i for each horse was calculated using the inverse cumulative distribution function, $z_i = \Phi^{-1}[p_n(i)]$ for the normal model. These scores predicted the probabilities of horses finishing in specific positions. Both models showed favorite-long shot bias, with the normal model performing better. This bias might stem from assuming independence between horse performances or the specific distribution forms. Another model is the weighted probabilistic model. Pudaruth et al. (2013) [3] developed a weighted probabilistic model for predicting horse racing outcomes by assigning calculated weights to various factors like jockey performance, new horses, and previous performance. However, the importance of each factor is predetermined, so this does not look at trends or new horses that come into later races. For example, the factor value for the performance of a jockey is determined by total wins over total mounts. Additionally, this way of calculating the factors does not account for correlation between factors, potentially overlooking significant interactions that could improve prediction accuracy. For example, a jockey's performance could be heavily dependent on the horse they are riding, or the weather on that day of the race. Pudaruth et al. [3] acknowledged these limitations and suggested that prediction accuracy can be increased by considering additional factors like age and equipment. They also proposed applying more advanced techniques such as neural networks, decision trees, and fuzzy logic to improve their model. We will build upon these insights in our approach.

3.2 Neural Networks

Neural networks make fewer assumptions than statistical models, learning patterns directly from data, which can capture complex non-linear relationships. Davoodi and Khanteymoori (2010) [2] explored this by using various neural network architectures and optimization algorithms to predict horse race outcomes. They trained a feed-forward neural network for each horse using horse/track features to predict finishing times, then ordered the horses based on these predictions. They experimented with different configurations and loss optimization algorithms, using sigmoid activation and mean square error (MSE) loss. The optimal configuration had two hidden layers with 5 and 7 neurons, respectively. Their model correctly predicted the first position in 39 races, the last position in 30 races, one horse's position in 43 races, two or more horses' positions in 33 races, and failed in 24 races out of 100. However, this paper approached the problem by training separate networks for each horse, hence assuming independence, which may have led to sub-optimal performance.

4 New Approach

We aim to mitigate the potential presented weaknesses by balancing our assumptions and viewpoint of races and race participants. Our approach will leverage the strengths of GB decision forests and regression models. With this setup, we hope the ensemble/meta model can learn the importance of each viewpoint during the training. We'll use historical data aggregation, focusing on past race results and performance data, avoiding the complexity of temporal dependencies from time series analysis.

5 Domain Vocabulary

For domain-specific vocabulary, see the table in the Appendix

5.1 GB decision forests

5.1.1 Data (GB decision forests)

We will be using the *Race Results* dataset to train our GB decision forests as they contain the necessary data of specific races and the results of them.
Unfortunately, there is a big variability in terms of number of observations with races of n participants in the dataset. This can be seen in this table:

Number of Participants (n)	Number of Observations (N)
4	4
6	169
7	248
8	250
9	177
10	131
11	85
12	30
13	13
14	7
17	1

We will only handle sizes with enough observations (*100-200* observations). This means our model can only handle races of size n where $6 \leq n \leq 10$. Specifically we will focus on $n = 7$ case.

5.1.2 Training (GB decision forests)

With the filtered dataset, we need to train 5 multi-output GB decision forests where decision forest d_i handles inputs with i horses with $i \in \mathbf{N}, 6 \leq i \leq 10$.
Each GB decision forest will output the predicted place of

the corresponding racer in the input:

$$d_i(x_1, x_2, \ldots, x_i, r) \rightarrow \{p_1, ..., p_i\}$$

where $p_j \in \{1, ..., i\}$ is the predicted place for racer $x_j \in \mathbb{R}^d$ (d features) and $r \in \mathbb{R}^D$ (D features) is a static race-specific vector for the input example.

5.1.3 Features

From general domain knowledge and relevant academic literature [3], the following features were selected in no particular order:
For static race specific features (r) - Race Distance, Run Up Distance, Weather, Race Surface, Track Condition, Race Month.
For racer specific features (x_i) - Age, Weight Carried, Dollar Odds, Equipment, Post Position, Trainer, Jockey

Feature types/encoding details available in the Appendix

5.1.4 Implicit Order Issue

A glaring issue arises in our training from the implicit order of the horse vectors $\{x_1, \ldots, x_i\}$. The model might learn to rely on the specific order of $x_1, \ldots, x_i$, causing discrepancies such as $d_i(x_1, x_2, \ldots, x_i) \neq d_i(x_1, x_3, x_2, x_4, \ldots, x_i)$ due to reorderings. We want the learner to understand that the inherent ordering should not affect its prediction.

Training on all permutations is impractical due to factorial growth, e.g., d_{10} with 10 inputs yields $10! = 3628800$ examples.

To mitigate this, we introduce a hyperparameter scaling function $\mathcal{K} : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ to control the subset size of permutations, defined as:

$$\mathcal{K}(x) = \left\lfloor x^{\frac{x}{3}} \log(x!) + x \right\rfloor$$

This non-linear scaling function grows similarly to $x!$ for smaller x while slowing down significantly for larger x. We hope taking a subset also helps with not overfitting and better generalization.

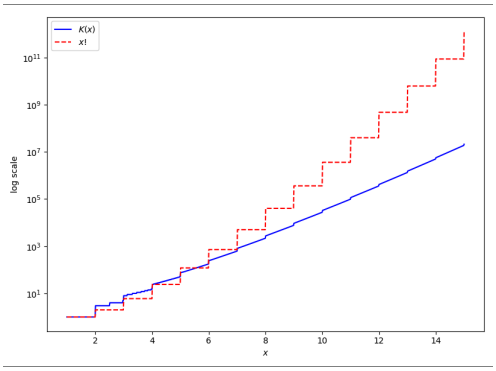


Figure 1: $\mathcal{K}$ and $x!$ comparison on logarithmic scale

5.1.5 Permutations Dataset Formalized

1. For any GB decision forest d_i where $i \in \mathbb{N}, 6 \leq i \leq 10$:
2. Define some data set for d_i by:

$$D_i = \{X_i^{(1)}, \ldots, X_i^{(N)}\}$$

where

$$X_i^{(j)} = [\mathbf{x}_1 \quad \mathbf{x}_2 \quad \cdots \quad \mathbf{x}_i \quad \mathbf{r_j}] \in \mathbb{R}^{1 \times (i \cdot d + D)}$$

is a data point with input vectors collapsed into a row vector and $N \in \mathbb{N}$ is the number of training examples for d_i.

3. We will create a new permutations training set T_i for d_i:
   (a) Define the total number of permutations to generate for each datapoint, $k \in \mathbb{N}$:

$$k = \mathcal{K}(i)$$

   (b) For each datapoint $X_i^{(j)}$: Define

$$p_j = \{(x_{z_1}, x_{z_2}, \ldots, x_{z_i}, r_j) \,|\, (x_{z_1}, \ldots, x_{z_i}) \in \mathit{Perm}(X_i^{(j)})\}$$

   as the ordered set of all permutations for $X_i^{(j)}$. Define $p'_j = \text{Sample}(p_w^{(j)}, \lfloor \frac{k}{i} \rfloor)$ as a random sample of size $\lfloor \frac{k}{i} \rfloor$ of p_j. Construct the permutation subset, P_j, for $X_i^{(j)}$:

$$P_j = \bigcup_{w=1}^{i} p'_w$$

   (c) Thus, the new training set is constructed by:

$$T_i = \bigcup_{j=1}^{N} P_j$$

5.1.6 Training Process

The training was done using gradient boosted trees from the *XGBoost* library , which leverages weak learners in producing a strong predictive model. XGBoost is favoured in many applications for its ability to capture complex patterns and usually performs better than traditional decision trees. To mitigate over-fitting on the small dataset, the following was done:

1. **Initial Split:** Split D_i into D_{i_1} (70% training/validation) and D_{i_2} (30% test)
2. **K-Fold Cross-Validation:** Perform 5-fold cross-validation on D_{i_1}:

$$D_{i_1}^{(k)} = \{D_{i_{1,train}}^{(k)}, D_{i_{1,val}}^{(k)}\}$$

3. **Permutations for Each Fold:** Create the permutations training set $T_i^{(k)}$ for each fold k
4. **Grid Search:** The following hyperparameter grid was searched:

Hyperparameter	Values
n_estimators	30, 50, 75, 100
max_depth	3, 5, 7
learning_rate	0.01, 0.05, 0.1, 0.2
subsample	0.8, 1.0
colsample_bytree	0.8, 1.0
reg_alpha	0, 0.01, 0.1, 0.5, 1, 2
reg_lambda	0, 0.01, 0.1, 0.5, 1, 2

5. **Model Training and Validation:** Train on $T_i^{(k)}$ and validate on $D_{i_{1,\text{val}}}^{(k)}$. Compute mean performance across all folds:

$$\mu = \frac{1}{K} \sum_{k=1}^{K} R(D_{i_{1,\text{val}}}^{(k)})$$

where $R(D_{i_{1,\text{val}}}^{(k)})$ is the rank-based accuracy, calculated by strict matching of the predicted ranks with the true ranks based on the average ranking of permuted outputs for the associated datapoint.

6. **Hyperparameters:** Select hyperparameters with the best mean performance:

$$\theta^* = \arg\max_{\theta} \mu$$

7. **Final Model:** The final model will be trained on D_{i_1} with the best hyperparameters θ^* and D_{i_2} will be saved for testing.

5.1.7 Results

Due to resource, page, and time limitations, we will only report the results and focus on the case: $i = 7$ (as it also contains second most datapoints with fewer permutations)

Parameter	Value	Prediction	Percentage (%)
colsample_bytree	0.8	Predict 1st place	46.82
learning_rate	0.05	Predict one in top 3	65.90
max_depth	5	Predict entire top 3	10.40
n_estimators	50	Predict all	2.31
reg_alpha	1	Predict at least 1	93.06
reg_lambda	0.1	**validation_score**	**0.23781**
subsample	1.0		

Table 1: Best Hyperparameters and K-Folds Training/Validation Metric Statistics

We won't report the test score for now in order to not introduce any biases in our ensemble model.

6 Linear Regression

Unlike GB decision forests, the regression model will provide a continuous output $y \in \mathbb{R}$ which represents the predicted finish time for the input horse in seconds and assumes conditional independence ie. $y_i \perp y_j \mid x_i, x_j \quad \text{for} \quad i \neq j$
We will define this model by:

$$l(x, r) \to \mathbb{R}^+$$

where $x \in \mathbb{R}^d$ is a racer vector and $r \in \mathbb{R}^D$ is the static vector.

6.1 Data (Linear Regression)

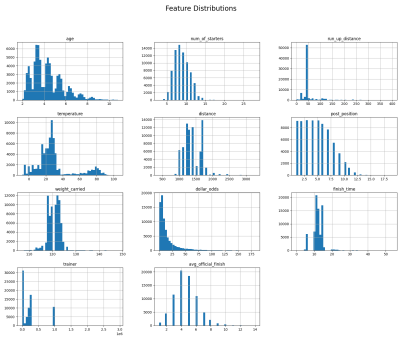


Figure 2: Past Performance Continuous Features Distribution

We will be using the *Past Performance* dataset as they contain all performance details about all the horses that ran a race in 2023. The dataset was preprocessed by removing race data before 2022, standardizing continuous values, and eliminating missing or constant columns.

6.1.1 Correlation (Past Performance Data)

As the goal of the regression model is to predict the finishing time, we will focus on correlation with respect to the finish time.

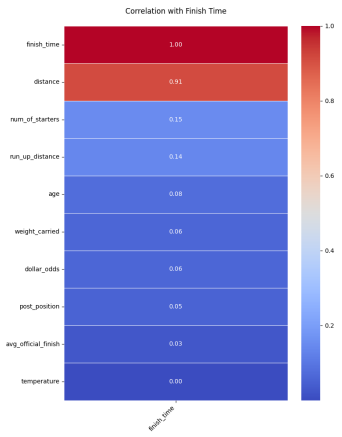


Figure 3: Continuous Data Correlation With Finish Time

From the correlation analysis, distance has the highest positive correlation (0.91) with finish time. Surprisingly, number of starters has the second highest correlation (0.15), *supporting our thesis that horse performance may not be independent of other competitors.*

6.2 Training (Linear Regression)

6.2.1 Features

From our correlation analysis, general domain knowledge, and relevant academic literature[1], the following features were selected in no particular order:

Feature Name	Type	Encoding / Details
Distance	Continuous	standardized to meters
Run up distance	Continuous	standardized to meters
Age	Continuous	Encoded as double (no rounding)
Weight Carried	Continuous	in pounds
Odds	Continuous	standardized to dollars
Number Of Starters	Continuous	
Jockey	Categorical id	encoded as one hot vector
Trainer	Categorical id	encoded as one hot vector
Post Position	Categorical	encoded as one hot vector
Equipment	Categorical	encoded as one hot vector
Sex	Categorical	encoded as one hot vector
Track Condition	Categorical	encoded as one hot vector
Scratched	Binary	binary value
Race Month	Categorical	encoded as one hot vector
Weather	Categorical	encoded as one hot vector
Medication	Categorical	encoded as one hot vector

Table 2: Feature to Type and Encoding table

Continuous features were log-transformed to stabilize variance and reduce outliers while minimizing extensive data shifts caused by stricter normalization techniques.

6.2.2 Configuration

Using the scikit-learn library and sparse matrices, we trained the model with L_2 regularization (Ridge regression) and experimented with 1^{st} degree and 2^{nd} degree feature mapping. We avoided higher degree polynomials due to the high dimension of our input space which caused very large polynomial expansions and increased overall complexity. 6

6.2.3 Test Split

Our dataset was partitioned into a $60\% - 20\% - 20\%$ split for the training, validation, and test sets respectively.

6.2.4 Linear Regression Training Results

From the figure and finer analysis, the following optimal hyper-parameters were found:

Hyper-Parameter	Value
Degree of Feature Mapping	2
α	1.733
Set	**MSE**
Validation Set	0.3710
Test Set	0.4006

Table 3: Optimal Hyper-Parameters and MSE

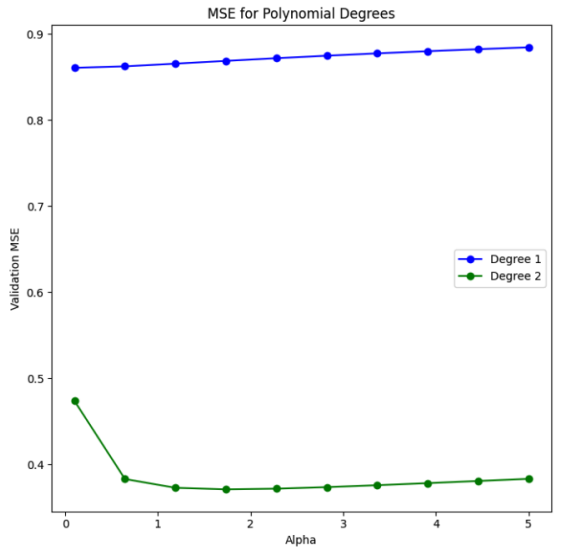


Figure 4: Validation MSE For Difference Alphas and Mappings

7 Ensemble Model Attempt

To learn the importance of each perspective/model, weights α and β will be assigned to each. The goal will be predicting the finish ordering. Note, due to page constraints, we will only cover a brief summary, but the full detailed process will be written in section 11.

7.1 Setup

Define $D = d_n(x_1, ..., x_n, r)$ (GB decision forest predicted raw output (without argsort) for datapoint) and $R = \{l(x_1, r), ..., l(x_n, r)\}$ (regression model finish times for racers in a datapoint).
We define the pairwise loss function of a single example w for a pair of racers i, j, where $i \neq j$ by:

$$\mathcal{L}_{ij}^{(w)} = \log\left[1 + \exp\left(-(t_i^{(w)} - t_j^{(w)})(y_i^{(w)} - y_j^{(w)})\right)\right]$$

where:

$$y_i = \alpha_i D_i + \beta_i R_i, \quad y_j = \alpha_j D_j + \beta_j R_j,$$

and t_i, t_j are the true rankings.
The loss for a single example is all the pair losses:

$$\mathcal{L}^{(w)} = \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \mathcal{L}_{ij}^{(w)}$$

For our cost function, we added l1 regularization for α, β:

$$\mathcal{J} = \frac{1}{N} \sum_{w=1}^{N} \sum_{j=1}^{n} \sum_{i=1, i\neq j}^{n} \mathcal{L}_{ij}^{(w)} + \frac{\lambda_1}{n} \sum_{i=1}^{n} |\alpha_i| + \frac{\lambda_2}{n} \sum_{i=1}^{n} |\beta_i|$$

We will use gradient descent to find optimal α, β

7.2 Training

First split the dataset D_i into D_{i1}, D_{i2}. Then perform 5 fold cross validation on D_{i1} using decision tree model d. Let $\hat{Y}_{\text{oof}}$ represent all the out of fold predictions from this process. We will combine $\hat{Y}_{\text{oof}}$ to form an unbiaset dataset of predictions and split this dataset into training and validation sets. Predictions from the regression model is used, where racers x_i and static vector r are extracted and used as inputs. The ensemble model e will be trained on the training portion of this dataset. Finally, train the base models d on the entire D_{i_1} and generate predictions for the ensemble model: $e = \text{train\_ensemble}(d(D_{i_1}, X_{i_1}))$

7.3 Results

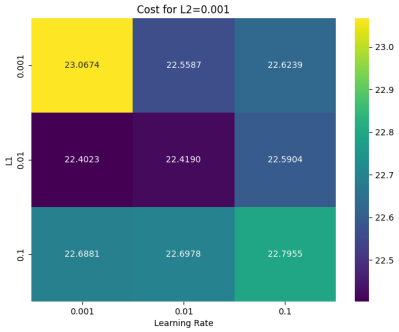


Figure 5: Weighted Ensemble Validation Cost Training Result

From our parameters grid search, our best parameters were $\lambda_1 = 0.01, \lambda_2 = 0.001, \eta = 0.01$
The performance of the ensemble model was lackluster to our expectations. Even with normalization techniques, like min-max normalization and Z-score normalization. The best validation score (validation score being the mean of percentage matched for validation examples) was 0.1755102 with min-max normalization on the input model outputs. Below are the validation statistics:

Prediction Metric	Stats Regression	Stats GB Forest	Stats Ensemble
Predict 1st place	22.86	31.43	**11.43**
Predict one in top 3	31.43	51.43	**28.57**
Predict entire top 3	0.00	2.86	**0.00**
Predict all	0.00	0.00	**0.00**
Predict at least 1	60.00	77.14	**80.00**

Table 4: validation statistics for models in %

7.4 Conclusion (Ensemble)

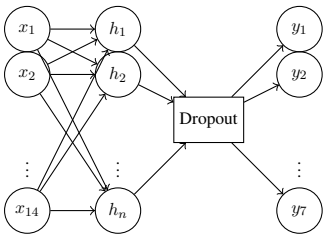
The results may suggest the limited expressive power of simply having a linear combination of weights α and β. The model's simplicity in addition to the small training set may have caused it to fail at capturing the complex non-linear relationships required for better predictions.

8 (Meta-Learner Attempt) Neural Network

From the pitfalls of the last attempt, we aim to capture more complex non-linear relationships in order to give better predictions by using a neural network as a meta-learner that takes in the two base model's outputs while optimizing with respect to the true ordering.

8.1 Setup (Meta Learner)

We will use the same dataset as the ensemble model D_{i_1} for training and D_{i_2} for validation. Due to the small training set, we found a two-layer architecture with non-linear activation functions and dropout to be the best configuration. We will use the architecture below and explore $dropout\_rate$ and $num\_neurons$ in the hidden layer as our parameters:



8.2 Training

During training, we explored the same normalization techniques as the ensemble model and found that min-max normalization also resulted in the best performance due to it's ability to balance both base-model contributions.

8.2.1 Loss Function (Hinge Rank Loss) and Activator Function (Leaky ReLU)

We experimented with various pairwise loss functions, but settled on *Hinge Rank Loss*. Loss functions without margin enforcement led to worse performance, as the model exploited small predictions without learning the importance of the magnitudes. Hinge Rank loss did a good job at preventing this through punishing close predictions.
We also used Leaky ReLU as it resulted in the best performance while reducing the chances of the vanishing gradient problem.

8.3 Neural Network Results

The following parameter grid was searched as the values were reasonable for the data-size and architecture:

Hyperparameter	Values
Learning Rates	0.0007, 0.0008, 0.0009
Number of Neurons	6, 7, 8
Dropout Rates	0.05, 0.1, 0.15
Num Epochs	80, 100, 120

Table 5: Neural Network Hyperparameter Grid Search

Best configuration:
$\eta = 0.0008$, $num\_neurons = 6$, $dropout\_rate = 0.05$ with a validation score of 0.20816326530612245.

8.3.1 Validation Results

Metric	Regression (%)	GB Forest (%)	Meta NN (%)	Ensemble (%)
Predict 1st place	22.86	31.43	34.29	11.43
Predict one in top 3	31.43	51.43	60.00	28.57
Predict entire top 3	0.00	2.86	2.86	0.00
Predict all	0.00	0.00	0.00	0.00
Predict at least 1	60.00	77.14	71.43	80.00

Table 6: Validation Set Statistics for Regression, GB Forest, NN, and Ensemble Models

9 Final Test Set Results For All Models

With all the models trained, the test set results are reported below **(74 races predicted)**:

Metric	Regression (%)	GB Forest (%)	NN (%)	Ensemble (%)
Predict 1st place	14.67	30.67	20.00	20.00
Predict one in top 3	36.00	57.33	36.00	44.00
Predict entire top 3	0.00	2.67	1.33	2.67
Predict all	0.00	1.33	0.00	0.00
Predict at least 1	69.33	84.00	60.00	76.00

Table 7: Test Set Statistics for Regression, GB Forest, NN, and Ensemble Models

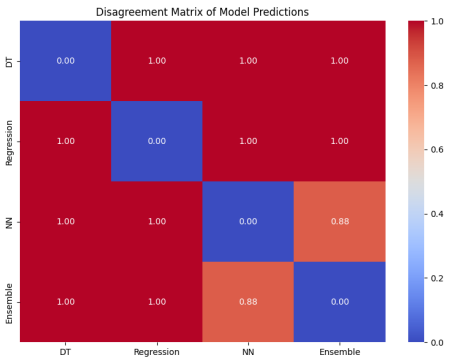


Figure 6: Test Disagreement Matrix

10 Final Reflection / Conclusion

The task of predicting the exact finishes of horse races proved to be challenging, especially with the small dataset of 248 examples for races with 7 racers. From our best model, the gradient boosted decision forest, it performed on par or worse than previous works but performs well on some metrics.
Even with many controls in place, the neural network showed signs of overfitting, as it had a higher validation score than the ensemble model but under performed worse on the test set. This may explain why the simpler ensemble model performed better on the test set as it generalized better with its simpler weights configuration. Notably, the regression model, which was trained on the independence assumption and past performance data, performed the worse. We believe this could have been either caused by the poor predictive nature of past performance data or that the independence assumption failed to capture the complex interactions between horses in a race. On the other hand, the gradient boosted decision forest model outperformed all other models, particularly because of its ability to generalize well on the small set of race results through our permutations approach while not following with the independence assumption. With its strong 1st place and top three prediction performance, it is clear it has a good understanding of what a winning horse looks like and its relative performance to other horses in the same race.

10.1 Evaluation (GB Decision Forest)

The performance of the GB decision forest model can be informally compared to other work, though it's important to consider that the datasets might differ significantly due to variations in race tracks, such as differences in humidity, elevation, or climate. For instance, compared to the neural network by Davoodi and Khanteymoori (2010) [2], which achieved a higher accuracy in predicting the first position (39% vs. 30.67%), the GB decision forest appears to have a better grasp of racer interactions, as it performed stronger in general ranking predictions. Specifically, the GB decision forest accurately predicted at least one racer 84% of the time, compared to 76% for the neural network and performed particularly well in predicting at least 1 racer in the top 3. However, these comparisons should be treated with caution, as they might be skewed by differences in reporting metrics and test set sizes; the neural network was tested on 100 data points, while the GB decision forest was tested on only 74.

10.2 Limitations

The disagreement matrix highlights a critical issue with the neural network and ensemble models. It's clear that there's a lot of disagreement between the regression and decision tree models. With such a small dataset, both high level models may have struggled to determine which model to trust or lean on, leading to poor generalization for both ensemble and neural network models. The neural network and ensemble models showed some agreement, indicating they might have partially identified which inputs to prioritize. However, the small dataset may have limited their ability to consistently make accurate decisions, leading to under-performance for both.

10.3 Looking Forwards / Future Improvements

Several steps could be taken to improve our implementations. Firstly, expanding the dataset would provide more data for the models to learn off on, so utilizing multiple years may be beneficial (we were limited to only the 2023 dataset). Additionally, with more domain knowledge and better feature engineering, performance may improve as most of our features were selected in a non-exhaustive way. Finally, while exploring other architectures could be beneficial, we still favour the meta neural network for its flexibility and potential to excel on larger datasets. For smaller datasets, we think XGBoost alone excels.

References

M. M. Ali, "Probability models and horse racing," *Journal of Applied Statistics*, vol. 25, no. 2, pp. 171–186, Mar. 1998. Available: https://www.tandfonline.com/doi/abs/10.1080/02664769823205

A. Khanteymoori and M. Hashemi, "Horse racing prediction using artificial neural networks," in *Proc. 10th Int. Conf. Intell. Syst. Design Appl.*, Cairo, Egypt, Nov. 2010. Available: https://www.researchgate.net/profile/Alireza-Khanteymoori/publication/228847950_Horse_racing_prediction_using_artificial_neural_networks/links/53fc54590cf2dca8ffff0df8/Horse-racing-prediction-using-artificial-neural-networks.pdf

S. Pudaruth, N. Medard, and Z. B. Dookhun, "Horse racing prediction at the Champ de Mars using a weighted probabilistic approach," *Int. J. Comput. Appl.*, vol. 72, no. 5, pp. 39–42, May 2013. Available: https://d1wqtxts1xzle7.cloudfront.net/53357232/pxc3889048-libre.pdf

Additional Information

11 Putting It All Together (Ensemble)

To learn the importance of each perspective/model, weights α and β will be assigned to each.

11.1 Dataset

The *Race Results* dataset will be used again as it contains results of races with participants.

11.2 Objective

Define datapoint, $x_1, ..., x_n \in \mathbb{R}^d$ and $r \in \mathbb{R}^D$ (same as GB decision forest definition). So, $(x_1, ..., x_n, r_j) \in X_j$ where the dataset of size $N \in \mathbb{N}$ is defined by $S = \{X_1, ..., X_N\}$. The goal will be the same as the GB decision forest's objective: predicting the finish ordering.

11.3 Setup

Ensemble Loss

Define $D = d_n(x_1, ..., x_n, r)$ (GB decision forest predicted raw output (without argsort) for datapoint) and $R = \{l(x_1, r), ..., l(x_n, r)\}$ (regression model finish times for racers in a datapoint).
We define the pairwise loss function of a single example w for a pair of racers i, j, where $i \neq j$ by:

$$\mathcal{L}_{ij}^{(w)} = \log\left[1 + \exp\left(-(t_i^{(w)} - t_j^{(w)})(y_i^{(w)} - y_j^{(w)})\right)\right]$$

where:

$$y_i = \alpha_i D_i + \beta_i R_i, \quad y_j = \alpha_j D_j + \beta_j R_j,$$

and t_i, t_j are the true rankings.
We chose this loss function, based on the logistic pairwise loss, as it is well-behaved (convex and smooth) and naturally penalizes incorrect rankings based on the true ranking. The loss for a single example is all the pair losses:

$$\mathcal{L}^{(w)} = \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \mathcal{L}_{ij}^{(w)}$$

So, the cost function for the dataset is:

$$\mathcal{J} = \frac{1}{N} \sum_{w=1}^{N} \sum_{j=1}^{n} \sum_{i=1, i\neq j}^{n} \mathcal{L}_{ij}^{(w)} + \frac{\lambda_1}{n} \sum_{i=1}^{n} |\alpha_i| + \frac{\lambda_2}{n} \sum_{i=1}^{n} |\beta_i|$$

11.3.1 Convexity:

Let $z = -(t_i - t_j)(y_i - y_j)$ and $f(z) = \log(1 + \exp(z))$. The first and second derivatives are:

$$f'(z) = \frac{\exp(z)}{1 + \exp(z)} = \sigma(z)$$

$$f''(z) = \sigma(z)(1 - \sigma(z))$$

where $\sigma(z)$ is the sigmoid function. Since $\sigma(z)(1 - \sigma(z)) \geq 0$, $f(z)$ is convex with respect to z. First-order partial derivatives of L_{ij}:

$$\frac{\partial L_{ij}}{\partial \alpha_k} = -(t_i - t_j) d_k \sigma(-(t_i - t_j)(y_i - y_j))$$

$$\frac{\partial L_{ij}}{\partial \beta_k} = -(t_i - t_j) r_k \sigma(-(t_i - t_j)(y_i - y_j))$$

where $\sigma(z) = \frac{\exp(z)}{1+\exp(z)}$. Second-order partial derivatives:

$$\frac{\partial^2 L_{ij}}{\partial \alpha_k^2} = (t_i - t_j)^2 d_k^2 \sigma(z)(1 - \sigma(z))$$

$$\frac{\partial^2 L_{ij}}{\partial \beta_k^2} = (t_i - t_j)^2 r_k^2 \sigma(z)(1 - \sigma(z))$$

The Hessian matrix H of L with respect to α and β is:

$$H = \begin{bmatrix} \frac{\partial^2 L}{\partial \alpha^2} & \frac{\partial^2 L}{\partial \alpha \partial \beta} \\ \frac{\partial^2 L}{\partial \beta \partial \alpha} & \frac{\partial^2 L}{\partial \beta^2} \end{bmatrix}$$

with positive second-order derivatives, ensuring H is positive semi-definite. Since L_{ij} is convex in α and β, and L is the sum of L_{ij}, L is convex with respect to α and β. Thus, L is convex.

11.3.2 Gradient Descent

Partials:
To perform gradient descent, we need the partials of L_{ij} with respect to $\alpha_i, \alpha_j, \beta_i, \beta_j$. These are given by:

$$\frac{\partial L_{ij}}{\partial \alpha_i} = \frac{(-(t_i - t_j)) \cdot d_i}{1 + \exp\left((t_i - t_j)(y_i - y_j)\right)}$$

$$\frac{\partial L_{ij}}{\partial \alpha_j} = \frac{(-(t_i - t_j)) \cdot (-d_j)}{1 + \exp\left((t_i - t_j)(y_i - y_j)\right)}$$

$$\frac{\partial L_{ij}}{\partial \beta_i} = \frac{(-(t_i - t_j)) \cdot r_i}{1 + \exp\left((t_i - t_j)(y_i - y_j)\right)}$$

$$\frac{\partial L_{ij}}{\partial \beta_j} = \frac{(-(t_i - t_j)) \cdot (-r_j)}{1 + \exp\left((t_i - t_j)(y_i - y_j)\right)}$$

Gradient Descent Derivations
For gradient descent, we compute the gradients of L with respect to each parameter. The gradients of the loss $\mathcal{L}$ for example w are:

$$\frac{\partial \mathcal{L}^{(w)}}{\partial \alpha_k} = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbb{I}[i \neq j] \cdot \frac{\partial \mathcal{L}_{ij}^{(w)}}{\partial \alpha_k}$$

$$\frac{\partial \mathcal{L}^{(w)}}{\partial \beta_k} = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbb{I}[i \neq j] \cdot \frac{\partial \mathcal{L}_{ij}^{(w)}}{\partial \beta_k}$$

Therefore,

$$\frac{\partial \mathcal{J}}{\partial \alpha} = \frac{1}{N} \sum_{w=1}^{N} \sum_{j=1}^{n} \sum_{i=1, i\neq j}^{n} \frac{\partial \mathcal{L}_{ij}^{(w)}}{\partial \alpha} + \frac{\lambda_1}{n} \sum_{i=1}^{n} \frac{\alpha_i}{|\alpha_i|}$$

For $\alpha_i \neq 0$ And,

$$\frac{\partial \mathcal{J}}{\partial \beta} = \frac{1}{N} \sum_{w=1}^{N} \sum_{j=1}^{n} \sum_{i=1, i\neq j}^{n} \frac{\partial \mathcal{L}_{ij}^{(w)}}{\partial \beta} + \frac{\lambda_2}{n} \sum_{i=1}^{n} \frac{\beta_i}{|\beta_i|}$$

Using the gradients of $\mathcal{J}$ with respect to α_k, β_k, we update the weights α and β as follows:

$$\alpha_k^{(t+1)} \leftarrow \alpha_k^{(t)} - \eta \cdot \frac{\partial \mathcal{J}}{\partial \alpha_k^{(t)}}$$

$$\beta_k^{(t+1)} \leftarrow \beta_k^{(t)} - \eta \cdot \frac{\partial \mathcal{J}}{\partial \beta_k^{(t)}}$$

where η is the learning rate. To do gradient checking, we approximated the numerical gradient of α and β as:

$$\frac{\partial \mathcal{J}}{\partial \theta} \approx \frac{\mathcal{J}(\theta + \epsilon) - \mathcal{J}(\theta - \epsilon)}{2\epsilon}$$

we got an average gradient difference of $10e^{-10}$ using epsilon = $10e^{-5}$, indicating correctness of the computed gradients

11.4 Building the Dataset for the Ensemble Model

11.4.1 Initial Split

- Split the dataset D_i into D_{i_1} and D_{i_2} with the exact same split as the GB decision forest.

11.4.2 K-Fold Cross-Validation on D_{i_1}

1. Perform 5-fold cross-validation on D_{i_1} using the decision tree model d:
   - Split D_{i_1} into K folds:

     $$\{D_{i_{1,train}}^{(k)}, D_{i_{1,val}}^{(k)}\} \quad \text{for} \quad k \in \{1, 2, \ldots, 5\}$$

   - For each fold k:
     - Train the model $d^{(k)}$ on $D_{i_{1,train}}^{(k)}$.
     - Generate out-of-fold predictions $\hat{y}_{i_{1,val}}^{(k)}$ on $D_{i_{1,val}}^{(k)}$:

       $$\hat{y}_{i_{1,val}}^{(k)} = d^{(k)}(D_{i_{1,train}}^{(k)})$$

   - Collect out-of-fold predictions for all folds:

     $$\hat{Y}_{\text{oof}} = \bigcup_{k=1}^{5} \hat{y}_{i_{1,val}}^{(k)}$$

11.5 Training Steps for the Ensemble Model

1. Combine $\hat{Y}_{\text{oof}}$ to form an unbiased dataset of predictions.
2. Split this dataset into $D_{\text{ens,train}}$ and $D_{\text{ens,val}}$ (80/20 split).
3. Use predictions from the regression model trained on the *Past Performance* dataset. Extract individual racers x_i and static vector r from each data point, and use the output $l(x_i, r)$ for each racer.
4. Train the ensemble model e on $D_{\text{ens,train}}$:

   $$e = \text{train}(D_{\text{ens,train}})$$

11.6 Final Training

- Train the base models d on the entire D_{i_1} and generate predictions for the ensemble model:

  $$e = \text{train\_ensemble}(d(D_{i_1}, X_{i_1}))$$

Appendix

Table 8: Domain Vocabulary

Vocabulary Name	Type	Explanation
Surface	Categorical	The surface of the race track (dirt, turf, synthetic)
Jockey	Categorical id	Individual human that rides the horse during races
Trainer	Categorical id	Cares for the horse: exercise, health, etc...
Owner	Categorical id	Financially owns the horse on paper
Medication	Categorical	Medications used by a horse for performance/health reasons
Equipment	Categorical	Equipment on a horse to enhance performance or control
Post Position	Categorical	The gate position of where the horse starts the race from
Scratched	Binary	Indicates if a horse was removed from a race
Weight Carried	Continuous	Weight of Jockey + Additional weights
Dollar Odds	Continuous	Potential payout on a 1 dollar bet on a horse (if it wins)
Run Up Distance	Continuous	Distance a horse travels before the official race timer starts

Static Race Specific Features (r) (GB Decision Forests)

Feature Name	Type	Encoding / Details
Race Distance	Continuous	standardized to meters
Run Up Distance	Continuous	standardized to meters
Weather	Categorical	label encoded
Race Surface	Categorical	label encoded
Track Condition	Categorical	label encoded
Race month	Categorical	label encoded

Racer Specific Features (x_i) (GB decision forests)

Feature Name	Type	Encoding / Details
Age	Continuous	Calculated as double (no rounding)
Weight Carried	Continuous	Standardized to pounds
Dollar odds	Continuous	
Equipment	Categorical	label encoded
Post Position	Categorical	label encoded
Trainer	Numerical id	Kept as integer
Jockey	Numerical id	Kept as integer