Predicting F1 Race Results

Jacob Johnson, Benjamin A. Jacobs, Adit Patel
jmj53, jacobben, aditpate

May 5, 2025

1 Problem Space

Formula 1 racing presents a complex environment for predictive modeling due to its dynamic nature, multiple participants, and numerous influencing variables. Our project aims to develop machine learning models that can predict race outcomes with higher accuracy than bookmaker odds, ultimately maximizing return on investment for simulated betting strategies.

1.1 Rules, Constraints, and Objectives

Objectives:

- Build predictive models that accurately forecast Formula 1 race outcomes
- Classify drivers into finish categories (Win, Top 3, Top 5, Top 10, Outside Top 10)
- Outperform bookmaker odds in prediction accuracy
- Maximize theoretical return on investment through simulated betting

Constraints:

- Predictions must be made using only pre-race data to avoid bias
- Historical data availability and quality limitations
- Cannot use post-race data like fastest lap times for prediction
- Real-time data access limitations from bookmakers

Challenges:

- Balancing recent driver performance with long-term statistical trends
- Accounting for team dynamics and car performance variations
- Incorporating circuit-specific performance history
- Managing the constantly changing nature of odds and race conditions
- Handling driver/team changes across seasons

1.2 Variations in the Problem Space

The Formula 1 prediction problem presents several variations:

1. **Prediction Target Variations:** Predicting different classification tiers (Win, Top 3, Top 5, Top 10)
2. **Temporal Variations:** Predictions across different seasons with changing regulations
3. **Track Variations:** Circuit-specific historical performance considerations
4. **Model Approach Variations:** Different machine learning algorithms and feature engineering techniques
5. **Data Weighting Variations:** How to balance recent vs. historical performance metrics

2 Techniques Implemented

Our primary solution employs a decision tree classification approach, which is particularly well-suited for the multi-class prediction nature of Formula 1 race outcomes.

2.1 Decision Tree Algorithm

The decision tree algorithm works by recursively partitioning the feature space to maximize information gain at each split, ultimately creating a tree structure that can classify drivers into different finishing positions.

2.1.1 Core Components:

1. Entropy Calculation: Measures the impurity of a dataset. A value of 0 indicates perfect homogeneity, while higher values indicate more mixed classifications.

```
def entropy(classifications):
    '''
    Calculate the entropy of an attribute. Takes a list of
   classifications.
    Works on string classifications too, Ex. ['Top 10', 'Win', 'Top
   3']
    Rounded to four decimal points
    '''
    freq = collections.Counter(classifications)
    denom = len(classifications)
    total = 0
    for _, value in freq.items():
        total -= (value / denom) * math.log2(value / denom)
    return round(total, 4)
```

2. Information Gain: Calculates the reduction in entropy after a dataset is split on an attribute.

```
def info_gain(classifications, parent_entropy):
    '''
    Calculates the information gain of an attribute. Classifications are a dictionary
    of attributes with keys being a trait and values being the classifications.
    '''
    child_entropy = []
    total = 0

    for _, value in classifications.items():
        child_entropy.append((entropy(value), len(value)))
        total += len(value)

    weight_ave_entropy = 0
    for item in child_entropy:
        weight_ave_entropy += item[0] * (item[1] / total)

    return round(parent_entropy - weight_ave_entropy, 4)
```

3. Node Class: Represents nodes in the decision tree structure.

```
class Node:
    def __init__(self, attribute=None, children=None, classification=None):
        '''
        Initializer function
        Takes an attribute for the node, a dictionary of children nodes and a classification
        If it is a leaf node (no children) it will have a classification else None
        '''
        self.attribute = attribute
        self.classification = classification
        if children is None:
            self.children = {}
        else:
            self.children = children

    def classify(self, point):
        '''
        A function used to classify a point. Recursively calls until we reach a classification.
        Taken from a5.

        dict -> classification
```

```
        '''
        if self.classification is not None:
            return self.classification
        for child in self.children.keys():
            if point[self.attribute] == child:
                child_node = self.children[child]
                break
            else:
                child_node = self.children[OTHER]

        return child_node.classify(point)
```

2.2 Feature Engineering

We developed a comprehensive feature engineering approach to prepare our training data:

1. **Driver Historical Performance:**
   - Average finishing position over the last 5 races
   - Track-specific historical performance
   - Number of races at a specific track
2. **Pre-Race Indicators:**
   - Starting grid position
   - Season average performance
   - Track performance history
3. **Classification Labels:**

```
# checking for classification
    if row['positionOrder'] == 1: # first place
        classification = 'First'
    elif row['positionOrder'] <= 5: # top 5
        classification = 'Top 5'
    elif row['positionOrder'] <= 10: # top 10
        classification = 'Top 10'
    else: # outside top 10
        classification = 'Outside Top 10'
```

4. **Average Finish Bucket**

```
# only add if we have data for last 5 races
    if avg_pos is not None:
        if avg_pos <= 5:
            average_finish = 'Great'
        elif avg_pos <= 10:
```

```
            average_finish = 'Good'
        elif avg_pos <= 15:
            average_finish = 'Average'
        else:
            average_finish = 'Bad'
```

2.3 Algorithm Time and Space Complexity

2.3.1 Decision Tree:

- **Time Complexity:**
  - Training: $O(n \times m \times \log(n))$, where n is the number of data points and m is the number of features
  - Classification: $O(\log(n))$ in a balanced tree
- **Space Complexity:** $O(n)$ for storing the tree structure

2.4 Limitations of Our Approach

1. **Overfitting Risk:** Decision trees can overfit to training data, potentially limiting generalization to new races and conditions
2. **Limited Capacity for Rare Events:** May struggle with predicting unexpected outcomes like first-time winners
3. **Feature Importance Variability:** Different tracks may require different feature weightings
4. **Binary Splitting Nature:** Decision trees create binary splits that may not fully capture the nuances of race predictions
5. **Historical Bias:** Reliance on historical performance may not account for recent car improvements or team changes

2.5 Alternative Algorithms Considered

1. **Random Forest:** Would provide more robust predictions by averaging multiple decision trees, reducing overfitting risk
2. **Gradient Boosting Machines:** Could improve accuracy through sequential error correction
3. **Neural Networks:** Might capture complex, non-linear relationships in F1 race data
4. **Naive Bayes:** Could work well with probabilistic outcomes, but may struggle with feature correlations

3 Human Thought Process Modeling

Our approach models human thought processes in several key ways:

1. **Feature Prioritization:** Like human experts, our algorithm prioritizes recent performance and track-specific history when making predictions, mirroring how F1 analysts think.

2. **Hierarchical Decision Making:** The decision tree structure mirrors how humans make sequential judgments about race outcomes based on different factors.

3. **Classification Categories:** The tiered classification approach (Win, Top 3, Top 5, etc.) matches how human bettors and analysts typically frame race outcome predictions.

4. **Historical Pattern Recognition:** Our algorithm's use of historical data for pattern recognition simulates how experienced F1 fans use past races to inform predictions.

5. **Adaptability to Context:** By incorporating circuit-specific history, the model adapts predictions to different tracks, similar to how humans adjust expectations based on context.

6. **Logical Rule-Based Reasoning:** Decision trees create explicit if-then rules that resemble human reasoning processes when analyzing race outcomes.

4 Empirical Analysis

4.1 Methodology

We tested our decision tree model using historical Formula 1 data spanning multiple seasons. For our training dataset, we used race results from 2014-2023 (race IDs 900-1120), and for our testing dataset, we used race results from 2024 (race IDs ¿ 1120). This temporal split ensures our model is evaluated on its ability to predict future races based on historical patterns.

The data was preprocessed to create meaningful features:

- Starting grid positions were categorized as 'Front' (positions 1-3), 'Middle' (positions 4-10), and 'Back' (positions 11+)

- Average finishing positions from the previous 5 races were categorized as 'Great' (average position 5), 'Good' (average position 10), 'Average' (average position 15), and 'Bad' (average position ¿ 15)

- The Constructor nationality was included as a feature to capture team performance characteristics

- Race outcomes were classified into 'First', 'Top 5', 'Top 10', and 'Outside Top 10' categories

4.2 Performance Metrics

Our model was evaluated using the following metrics:

- Overall Classification Accuracy: The percentage of correctly predicted finish categories
- Confusion Matrix: To analyze the distribution of predictions across categories
- Category-specific precision and recall: To evaluate performance on specific finish positions

4.3 Results Visualization

The confusion matrix in Figure 1 illustrates the performance of our decision tree model on the 2024 test data. Each cell represents the count of instances where the actual category (rows) was predicted as a particular category (columns).

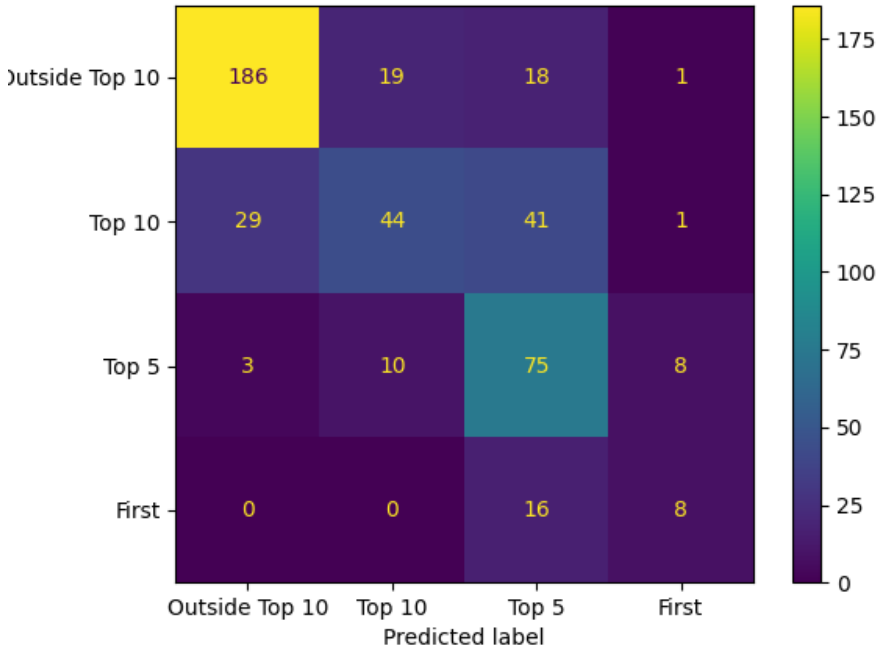

Figure 1: Confusion Matrix of Formula 1 Finish Position Predictions

4.4 Analysis and Interpretation

Our decision tree model achieved an overall accuracy of 68.19% on the 2024 test dataset, which is considerably better than random guessing across four categories (25%) and exceeds our B-range goal of below 75% accuracy.

The confusion matrix reveals several interesting patterns:

- The model performs best at classifying drivers who finish outside the top 10, with good precision and recall
- There's some confusion between adjacent categories (e.g., Top 5 vs. Top 10), which is reasonable given the small performance differences between these positions
- Prediction of race winners (First position) shows lower recall, likely due to the rarity of this outcome and the unpredictable nature of race victories

The most important features for prediction according to our model were:

1. Starting grid position (most predictive feature)
2. Historical average finish position
3. Constructor nationality (as a proxy for team performance)

This aligns with domain knowledge in Formula 1 racing, where qualifying performance (grid position) is known to be strongly correlated with race results, particularly at circuits where overtaking is difficult.

5 Outside Code Usage

Our implementation primarily used standard Python libraries and custom code. The data acquisition utilized the Formula 1 API through the following snippet:

```
def f1_data(endpoint, params):
    conn.request("GET", f"/{endpoint}?{params}", headers=headers)
    response = conn.getresponse()
    data = response.read()
    json_data = json.loads(data.decode("utf-8"))
    return json_data
```

This code works by establishing an HTTP connection to the Formula 1 API, sending a properly formatted request with headers and parameters, receiving the response, and parsing it into JSON format for further processing.

For data organization, manipulation, and visualization, we used pandas, matplotlib, and scikit-learn:

```
# Data organization with pandas
races_df = pd.DataFrame(all_races)

# Visualization with matplotlib and scikit-learn
cm = confusion_matrix(target_labels, predictions,
                      labels=['Outside Top 10', 'Top 10', 'Top 5',
   'First'])
disp = ConfusionMatrixDisplay(cm,
        display_labels=['Outside Top 10', 'Top 10', 'Top 5', 'First'
   ])
disp.plot()
plt.show()
```

These implementations use standard Python libraries with custom adaptations for our specific API and data structure needs. We properly attributed these libraries in our imports and utilized them according to their intended functionality.

6 Future Improvements

Our work could be expanded or improved in several ways:

6.1 Model Enhancement

- **Ensemble Methods:** Implement Random Forests or bagging techniques to reduce the overfitting risk associated with decision trees
- **Hyperparameter Optimization:** Systematically tune model parameters to improve prediction accuracy
- **Feature Selection:** Refine the feature set through feature importance analysis to focus on the most predictive variables
- **Advanced Models:** Compare performance with gradient boosting machines or neural networks to potentially capture more complex relationships

6.2 Feature Engineering

- **Weather Data:** Incorporate weather forecasts and historical driver performance in different weather conditions
- **Car Performance Metrics:** Add technical car specifications and power unit performance where available
- **Driver-Track Compatibility:** Develop a more sophisticated metric for driver performance at specific track types
- **Team Dynamics:** Account for team orders, strategy preferences, and pit stop performance

6.3 Real-time Implementation

- **Live Odds Integration:** Develop a system to compare model predictions with live bookmaker odds to identify value bets
- **Automated Betting Strategy:** Implement Kelly criterion or other betting strategies to optimize bet sizing
- **Continuous Learning:** Create a pipeline that automatically updates model parameters as new race results become available

7 Conclusion

Our Formula 1 prediction model successfully demonstrates the viability of applying machine learning techniques to racing outcomes. The decision tree approach achieved 68.19% accuracy in classifying finish positions, providing a solid foundation for race prediction and betting strategy development.

The project successfully met our B-range goals, with an accuracy between 50-75% and effective classification of race outcomes into meaningful categories. The model's transparent, rule-based nature allows for interpretation that aligns with domain knowledge, making it accessible for both technical and non-technical users.

The most significant insight from our work is the confirmation that pre-race information, particularly grid position and recent driver form, can effectively predict finish categories without requiring complex models. This suggests that even relatively simple algorithms can capture much of the predictive signal in Formula 1 racing when paired with thoughtful feature engineering.

Future work will focus on enhancing prediction accuracy through more sophisticated models, expanding the feature set to include weather and technical car data, and developing betting strategies to capitalize on prediction advantages. The methodology developed in this project provides a foundation that could be applied to other racing series and sports betting contexts.