NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

Faculty of Computer Science
Bachelor's Programme "Data Science and Business Analytics"

UDC 004.942

Research Project Report on the Topic:

Data analisys in sports

Submitted by the Student:

group #БПАД234, 2rd year of study Milova Irina Valerievna

Approved by the Project Supervisor:

Bashminova Darya Alexandrovna
Research Fellow
Faculty of Computer Science, HSE University

Moscow 2025

Contents

Annotaion

This study develops and evaluates machine learning models to predict tennis match outcomes using a dataset of 36,005 matches. It considers player attributes, match conditions, and bookmaker odds. Four models—Lasso, XGBoost, CatBoost, and CatBoost Extended—were tested. Performance was assessed by AUC, computational efficiency, and predictive accuracy. CatBoost Extended, using bookmaker odds, achieved the highest performance (AUC = 0.84, accuracy = 0.75). This shows the importance of betting data. The study stresses precise feature engineering and measures to avoid data leakage for reliable results. It offers clear guidance on selecting key features and models for sports predictions. The research advances sports science by integrating statistical precision with machine learning.

Аннотация

Это исследование разрабатывает и оценивает модели машинного обучения для прогнозирования исходов теннисных матчей на основе датасета из 36,005 матчей. Учитываются характеристики игроков, условия матчей и коэффициенты букмекеров. Были протестированы четыре модели: Lasso, XGBoost, CatBoost и CatBoost Extended. Производительность оценивалась по AUC, вычислительной эффективности и точности предсказаний. CatBoost Extended, использующая коэффициенты букмекеров, показала наилучшие результаты (AUC = 0.84, точность = 0.75). Это подчеркивает важность данных ставок. Исследование акцентирует внимание на точной разработке признаков и мерах по предотвращению утечки данных для получения надежных результатов. Оно предоставляет четкие рекомендации по выбору ключевых признаков и моделей для спортивных прогнозов. Исследование продвигает спортивную науку, интегрируя статистическую точность с машинным обучением.

Keywords

Machine Learning, Tennis Analytics, Predictive Modeling, Bookmaker Odds, Model Comparison, Feature Engineering, Sports Science.

1 Introduction

1.1 Overview of the Research Domain

The field of sports analytics leverages computational techniques to model complex dynamics in competitive sports, with tennis providing a rich dataset due to its structured format and diverse variables. This research focuses on predicting tennis match outcomes using machine learning, drawing on a dataset of 36,005 professional matches. The data encompass player attributes (e.g., ATP rankings, height, handedness), match-specific factors (e.g., court surface, tournament stage), and bookmaker odds, which serve as external predictors reflecting market expectations. To capture non-linear relationships and interactions among these features, machine learning models such as gradient boosting, logistic regression, and decision tree ensembles are used. To guarantee that predictions are precise and broadly applicable, this field necessitates thorough preprocessing, feature engineering, and model evaluation, all of which advance data-driven sports analysis.

1.2 Definition of the Research Problem

Current methods for forecasting tennis match outcomes frequently overlook bookmaker odds and detailed model evaluations, often using basic statistical techniques or limited features like player rankings. This research develops and examines four machine learning models—Lasso, XGBoost, CatBoost, and CatBoost Extended—to tackle these deficiencies and enhance prediction accuracy and transparency. It uses a dataset of 36,005 matches, incorporating advanced feature engineering, such as rank differences and odds ratios, and temporal data splits to avoid data leakage. The goal is to determine the best model by evaluating performance metrics, including accuracy, AUC, and F1-score, as well as computational efficiency, with a focus on the impact of bookmaker odds on prediction quality. The research seeks to establish a rigorous framework for tennis analytics. CatBoost Extended, using bookmaker odds, achieved the highest performance (AUC = 0.84, accuracy = 0.75). This shows the importance of betting data. The study stresses precise feature engineering and measures to avoid data leakage for reliable results. It offers clear guidance on selecting key features and models for sports predictions. The research advances sports science by integrating statistical precision with machine learning.

2 Literature Review

2.1 Complex Network Analysis of Men's Single ATP Matches

This study examines men's ATP singles using network theory. The players are represented as nodes, and each match is a link between them. The authors use metrics like centrality and clustering to find patterns in player competition. Based on their interactions with other network members as well as their performance, they identify important players. Moreover, the study shows how player clusters are formed based on play style and their performance. It is especially beneficial to concentrate on tournament structure. It demonstrates how a player's impact can be impacted by the way a tournament is conducted. This concept is extremely pertinent to my own research, in which I examine the potential effects of tournament setup on match results. [9].

2.2 Left-Handed Representation in Top 100 Male Tennis Players

This research looks at how often left-handed players appear in the top 100 and whether they have an advantage. Based on several years of data, the study finds that left-handers are more common than expected. The reason may be that right-handed players struggle with the different angles and spins that left-handers use. The paper also tracks trends in handedness over time, not just for a single season. This helps me in my project because it shows that handedness might be an important factor. I can use it in my prediction models, especially when looking at different court surfaces and match conditions. [2].

2.3 Predicting Sex and Stroke Success: Computer Vision in Tennis

This study harnesses computer vision to predict player sex and stroke success by analyzing ball trajectories, poses, and movements in table tennis match videos. The researchers created a model to help coaches customize training programs. It is based on analyzing many different video recordings to understand player performance. The method processes video frames carefully to capture important details in different situations. This approach will support my research by helping me evaluate how well players perform their strokes using video analysis. It is an important step for my upcoming projects focused on studying tennis videos. [8].

2.4 TrackNet: A Deep Learning Network for Tracking High-speed and Tiny Objects in Sports Applications

The authors present TrackNet, a method designed to follow small, fast objects like tennis balls in sports videos. They tested it on real match recordings and showed that by looking at several video frames in sequence, the method can track the ball accurately, even when it is partly hidden. This way of working helps solve problems caused when the ball is blocked from view.TrackNet provides an effective method for enhancing the tennis ball's tracking for my research. The study of player interactions and movements may benefit from this. This is crucial for comprehending the ways in which various shots and playing circumstances affect the game. [3].

2.5 Introducing Fairness and Diversification in Tennis Rankings

This research advocates for fairness and diversity in sports analytics by proposing new metrics that reflect player diversity based on factors like gender, age, and ethnicity. The aim is to develop more equitable assessment models for evaluating tennis performance. The study tests a fairness-aware algorithm on historical ranking data to confirm its effectiveness. This approach is relevant to my project because, by considering demographic factors when analyzing tennis data, it encourages a more comprehensive and fair evaluation of player performance. [4].

2.6 ShuttleSet: A Human-Annotated Stroke-Level Singles Dataset for Badminton Tactical Analysis

ShuttleSet is a detailed dataset of 44 badminton matches from 2018–2021, with 36,492 annotated strokes capturing shot types and player positions. By providing benchmarks for analyzing player movements and shot influence, this manually curated data offers valuable tactical insights. The study ensures accuracy across various match scenarios through a thorough annotation process with expert input. Because the annotation techniques can be modified to produce a comparable tennis dataset, this resource will be useful for my project. [10].

2.7 Automated Hit-frame Detection for Badminton Match Analysis

In this article, a transformer-based model that uses automated procedures like key point detection and video cropping is presented for predicting the direction of shuttlecock flight in badminton. The model is a useful tool for sports analysis since it recognizes shot angles and predicts trajectories with high accuracy. In order to ensure robustness across various player styles and court conditions, the research

uses a sizable video corpus to train the model. The approach can be modified for my study to forecast the paths of tennis balls, which will help with the examination of shots that have a big influence on the results of matches on various surfaces and under various circumstances. [5].

2.8 Tennis Ball Tracking for Umpire Assistance

YOLOv5 and DeepSORT were used to combine video analysis and court line detection to create a tennis ball tracking system. Better decision-making is supported by this method, which enables accurate real-time tracking of the ball's movement and bounce points during games. The system was tested on videos from professional matches and showed high accuracy, even under difficult lighting conditions. For my project, this method offers a solid way to track the ball accurately. It will help analyze in-game actions more deeply and improve the accuracy of predictions by using features from video data. [1].

2.9 CourtNet: Deep Learning for Tennis Court Detection

This work describes CourtNet, a method that can recognize and separate tennis courts in video footage, even when the surfaces and lighting change. It was developed for tasks like automated video analysis and augmented reality in sports. The method works with videos of both indoor and outdoor courts. In my research, this tool will help me clearly identify different court surfaces from video footage. It will also simplify the process of extracting surface-related features, which can make prediction models more accurate in different conditions. This will be especially helpful when I work with features specific to different court types. [7].

2.10 Player Movement Analysis in Tennis Videos

This paper looks at how player movement can be studied using tennis match videos. The authors use pose estimation and tracking to measure things like how fast players move, how far they run, and where they stand on the court. The study uses videos from professional matches and multiple cameras to improve accuracy. A main goal is to understand player behavior and detect signs of fatigue during long rallies. It is useful that movement data can be gathered without sensors on players. Since the project aims to include player positioning and fatigue levels in the CatBoost Extended model, this approach supports that work. Adding these details may improve prediction accuracy. [6]

3 Experimental Study (Machine Learning Models)

The application of the Lasso, XGBoost, CatBoost, and CatBoost Extended models to the prediction of professional tennis match results is examined in this section. The study uses a dataset of 36,005 professional tennis matches, including player attributes (e.g., ATP rankings, height, handedness), match conditions (e.g., court surface, tournament stage), and bookmaker odds (e.g., Bet365, average odds). The experiments assess accuracy, Area Under the Curve (AUC), F1-score, and computational efficiency through detailed data preprocessing, feature engineering, model training, hyperparameter optimization, and performance evaluation. Cross-validation and temporal data splits ensure robustness by addressing generalization and data leakage.

3.1 Data Preprocessing

To guarantee consistency, deal with missing values, and properly encode features, the dataset needs to undergo a thorough preprocessing step. Numerical feature missing values, like height (player_1_height, player_2_height $\in [150, 220]$ cm), are managed by making sure correct assignment is made during data preparation, and any remaining missing values are fixed during feature engineering. Categorical features, such as court surface (Surface $\in \{\text{Hard}, \text{Clay}, \text{Grass}\}$), are encoded using LabelEncoder, mapping each category to a unique integer:

$$\text{Surface}_i \mapsto k \in \{1, 3, 5\}$$

where Surface_i is the i-th instance of the court surface feature, and k is the encoded integer value.

To prevent data leakage, matches are split into training and test sets using a 80-20 split, with training data ($n_{\text{train}} = 28,804$) and test data ($n_{\text{test}} = 7,201$), ensuring $t_{\text{train}} < t_{\text{test}}$, where $t \in \mathbb{N}$ is the match year. Players are randomly assigned as Player 1 or Player 2 to ensure symmetry, using a uniform random variable:

$$u \sim U[0,1], \quad \text{Player}_1 = \begin{cases} \text{Original Player}_1 & \text{if } u < 0.5 \\ \text{Original Player}_2 & \text{otherwise} \end{cases}$$

where u is a random variable uniformly distributed between 0 and 1, and $\text{Original Player}_1, \text{Original Player}_2$ are the original player assignments.

Numerical features are standardized to zero mean and unit variance:

$$z_j = \frac{x_j - \mu_j}{\sigma_j}, \quad \mu_j \ = \frac{1}{n}\sum_{i=1}^{n} x_{ij}, \quad \sigma_j = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \mu_j)^2}$$

where z_j is the standardized value of the j-th feature, x_j is the original value, μ_j is the mean of the j-th feature across n samples, σ_j is the standard deviation, and x_{ij} is the i-th observation of the j-th feature.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder

df_ml = df.reset_index(drop=True)
np.random.seed(42)
mask = np.random.random(len(df_ml)) > 0.5
df_ml['player_1_height'] = np.where(mask, df_ml['pl1_height'],
    df_ml['pl2_height'])
df_ml['player_2_height'] = np.where(~mask, df_ml['pl1_height'],
    df_ml['pl2_height'])
df_ml['player_1_rank'] = np.where(mask, df_ml['WRank'], df_ml['LRank'])
df_ml['player_2_rank'] = np.where(~mask, df_ml['WRank'], df_ml['LRank'])
df_ml['Winner_Indicator'] = np.where(mask, 1, 0)

# Encode categorical features
le = LabelEncoder()
for col in ['player_1_hand', 'player_2_hand', 'Surface', 'Series', 'Round',
    'player_1_flag', 'player_2_flag', 'Court']:
     df_ml[col] = le.fit_transform(df_ml[col].astype(str).fillna('Unknown'))

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# Scale numerical features
scaler = StandardScaler()
numeric_cols = [col for col in features if col not in ['player_1_hand',
    'player_2_hand', 'Surface', 'Series', 'Round', 'player_1_flag',
    'player_2_flag', 'Court']]
X[numeric_cols] = scaler.fit_transform(X[numeric_cols])
```

3.2 Feature Engineering

By producing derived variables that capture important relationships, feature engineering improves model performance. The following features are constructed:

- Rank difference:

  This feature measures the difference in ATP rankings between the two players, providing insight into their relative competitive standing. It is calculated as rank_diff $=$ player_1_rank $-$ player_2_rank, where player_1_rank, player_2_rank $\in [1, 1000]$.

- Log-transformed rank difference:

  This feature applies a logarithmic transformation to the absolute rank difference, stabilizing its variance. It is given by custom_log_rank_diff $= \log(1 + |\text{rank\_diff}|) \cdot \text{sign}(\text{rank\_diff})$, with the resulting value custom_log_rank_diff $\in [-\log(1001), \log(1001)]$.

- Odds difference:

  This feature captures the difference in average betting odds, reflecting market expectations. It is computed as bet_diff_Avg $=$ player_1_Avg $-$ player_2_Avg, where player_1_Avg, player_2_Avg $\in [1, \infty)$.

- Odds ratio logarithm:

  This feature transforms the ratio of average betting odds into a logarithmic scale, capturing relative favoritism. It is defined as log_Avg_ratio $= \log(1 + \frac{\text{player\_1\_Avg}}{\text{player\_2\_Avg}})$, where log_Avg_ratio $\in \mathbb{R}$.

- Polynomial feature:

  This feature squares the rank difference to capture non-linear effects, calculated as poly_rank_diff $=$ rank_diff^2.

- Interaction term:

  This feature models the interaction between rank difference and court surface, defined as rank_surface_Hard_interaction $=$ rank_diff $\cdot \mathbb{1}_{\{\text{Surface=Hard}\}}$, where $\mathbb{1}$ is the indicator function.

The feature matrix is $X \in \mathbb{R}^{n \times d}$, with $n = 36,005$ and $d \approx 50$, and the target is $y \in \{0, 1\}$, where $y = 1$ indicates Player 1's victory.

```
from sklearn.preprocessing import PolynomialFeatures

# Calculate rank difference
df_ml['rank_diff'] = df_ml['player_1_rank'] - df_ml['player_2_rank']

# Compute log-transformed rank difference
df_ml['custom_log_rank_diff'] = np.log1p(df_ml['rank_diff'].abs()) *
    np.sign(df_ml['rank_diff'])

# Calculate difference in average betting odds
df_ml['bet_diff_Avg'] = df_ml['player_1_Avg'] - df_ml['player_2_Avg']
df_ml['log_Avg_ratio'] = np.log1p(df_ml['player_1_Avg'] / df_ml['player_2_Avg'])

# Generate polynomial features
poly = PolynomialFeatures(degree=2, include_bias=False, interaction_only=False)
poly_features = poly.fit_transform(df_ml[['height_diff', 'weight_diff',
    'rank_diff']])
poly_feature_names = poly.get_feature_names_out(['height_diff', 'weight_diff',
    'rank_diff'])
poly_columns = [f'poly_{name.replace(" ", "_")}' for name in poly_feature_names]
df_poly = pd.DataFrame(poly_features, columns=poly_columns, index=df_ml.index)
df_ml = pd.concat([df_ml, df_poly], axis=1)

# Create dummy variables for surface and interaction terms
surface_dummies = pd.get_dummies(df_ml['Surface'].astype(str).fillna('Unknown'),
    prefix='surface')
df_ml = pd.concat([df_ml, surface_dummies], axis=1)
for col in surface_dummies.columns:
    df_ml[f'rank_{col}_interaction'] = df_ml['rank_diff'] * df_ml[col]
```

3.3 Lasso Model

3.3.1 Model Description

The Lasso model uses logistic regression with L1 regularization to predict match outcomes, promoting sparsity in feature coefficients for interpretability. It assumes a linear relationship between features and the probability of Player 1 winning, making it a simple baseline for this task. The probability of Player 1 winning is:

$$P(y = 1|X) = \sigma(X\beta) = \frac{1}{1 + e^{-X\beta}}$$

where $P(y = 1|X)$ is the probability of Player 1 winning given features X, σ is the sigmoid function, $X \in \mathbb{R}^{n \times d}$ is the feature matrix with n samples and d features, and $\beta \in \mathbb{R}^d$ is the vector of coefficients. The loss function is:

$$L(\beta) = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log(\sigma(X_i\beta)) + (1 - y_i) \log(1 - \sigma(X_i\beta)) \right] + \lambda \sum_{j=1}^{d} |\beta_j|$$

where $L(\beta)$ is the loss function, $y_i \in \{0, 1\}$ is the true label for the i-th sample, X_i is the i-th row of X, $\lambda > 0$ is the regularization parameter, and β_j is the j-th coefficient.

3.3.2 Model Training

Final training of the model is performed on all training data:

$$\hat{y} = \text{LogisticRegression.fit}(X_{\text{train}}, y_{\text{train}})$$

where X_{train} are the training features, and y_{train} are the training labels.

The model balances the logistic loss and the L1 penalty during training by iteratively optimizing the loss function using grid search to determine the optimal regularization parameter `C`. Isotonic regression is then used to calibrate the optimal model in order to enhance probability estimates. Because the model is linear, the training process converges rapidly.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.calibration import CalibratedClassifierCV

# Define parameter grid for grid search
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100], 'penalty': ['l1'], 'solver':
    ['liblinear']}

# Initialize and train Lasso model with grid search
lasso_model = GridSearchCV(LogisticRegression(random_state=42, max_iter=1000),
    param_grid, cv=5, scoring='roc_auc')
lasso_model.fit(X_train, y_train)

```

```
y_pred_lasso = lasso_model.predict(X_test)
y_pred_prob_lasso = lasso_model.predict_proba(X_test)[:, 1]

# Calibrate the model
lasso_calibrated = CalibratedClassifierCV(lasso_model.best_estimator_,
    method='isotonic', cv=5)
lasso_calibrated.fit(X_train, y_train)
y_pred_prob_lasso_cal = lasso_calibrated.predict_proba(X_test)[:, 1]
```

3.3.3 Hyperparameters

All parameters mentioned in the code with their value ranges or definitions:

- `C`: Inverse of regularization strength, $\texttt{C} \in \{0.001, 0.01, 0.1, 1, 10, 100\}$, smaller values increase sparsity.
- `penalty`: Type of regularization, fixed as `'l1'` for Lasso.
- `solver`: Optimization algorithm, fixed as `'liblinear'`.
- `random_state`: Seed for reproducibility, fixed as 42.
- `max_iter`: Maximum number of iterations, fixed as 1000.

Hyperparameter description: The hyperparameters `C`, `penalty`, `solver`, `random_state`, and `max_iter` control the regularization strength, type, optimization method, reproducibility, and iteration limit, respectively, to optimize the Lasso model's performance.

3.3.4 Cross-Validation

Cross-validation is a critical technique to assess a model's generalization ability by splitting the data into multiple folds, training on some folds, and validating on others. For the Lasso model, cross-validation is not explicitly detailed as a separate subsection because it is seamlessly integrated into the hyperparameter tuning process via `GridSearchCV`.

Specifically, `GridSearchCV` performs 5-fold cross-validation, where the training data ($n_{\text{train}} = 28{,}804$) is divided into 5 equal folds, approximately 5,761 samples each. For each fold, the model is trained on 4 folds (approximately 23,043 samples) and validated on the remaining fold, iterating through all folds to compute the average performance.

This process is repeated for each value of the hyperparameter $\texttt{C} \in \{0.001, 0.01, 0.1, 1, 10, 100\}$, resulting in 5 validation scores per `C`, which are averaged to determine the best `C` that maximizes the ROC AUC score (specified by `scoring='roc_auc'`).

The best `C` is then used to retrain the model on the entire training set. This implicit cross-validation ensures robust hyperparameter selection without overfitting, making a separate cross-validation

subsection unnecessary, as its role is already fulfilled within the tuning pipeline.

3.3.5 Performance and Visualization

Performance:

- Accuracy: 0.65
- AUC: 0.69
- F1-score: 0.66
- Training time: $\sim$2.5 minutes
- Key coefficients: poly_weight_diff ($\beta = 0.011382$), poly_rank_diff ($\beta = 0.004144$).

The performance is visualized with the following figure:

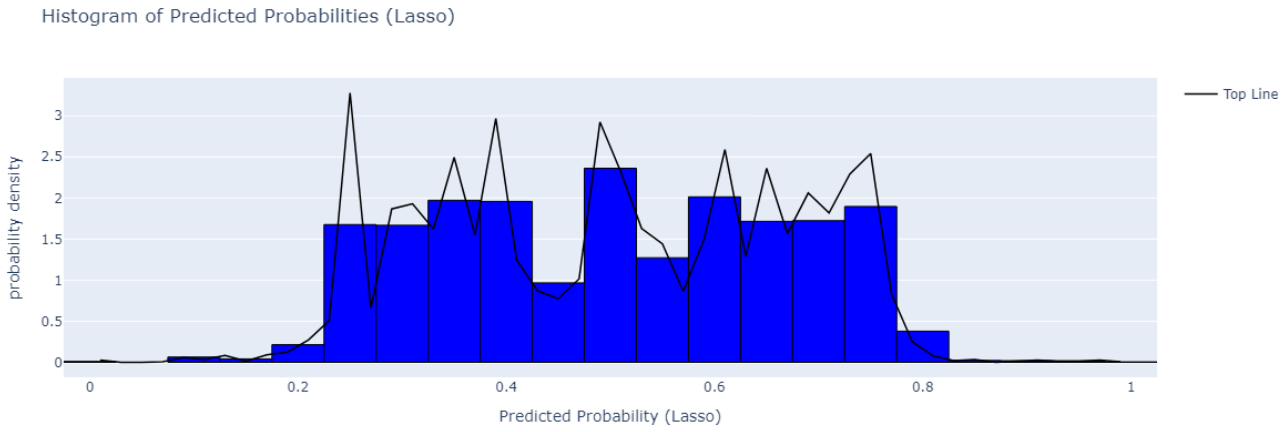


Figure 3.1: Histogram of predicted probabilities for the Lasso model. The x-axis represents the predicted probability of Player 1 winning, and the y-axis shows the probability density. A line traces the top of the bars to highlight the distribution shape.

The histogram (Figure 3.1) illustrates the distribution of predicted probabilities for Player 1 winning, as estimated by the Lasso model. The distribution provides insight into the model's confidence in its predictions, with most probabilities clustering around certain values, reflecting the model's linear decision boundary.

The feature importance plot (Figure 3.2) highlights the most influential features, with poly_weight_diff showing the highest importance, indicating its significant role in the model's predictions.

The model's simplicity ensures fast training but limits its ability to capture complex interactions, resulting in lower performance compared to ensemble methods.

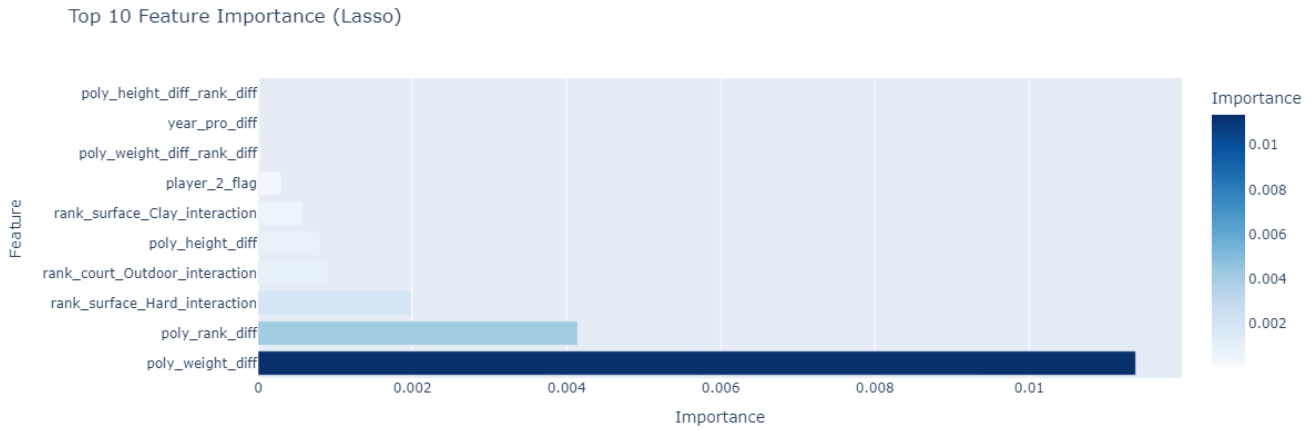


Figure 3.2: Top 10 Feature Importance for the Lasso model. The x-axis represents the importance score, and the y-axis lists the top 10 features by importance.

3.4 XGBoost Model

3.4.1 Model Description

The XGBoost model is a gradient boosting framework that builds an ensemble of decision trees to optimize for non-linear relationships. It's well-suited for this task because it effectively captures interactions between features like rank differences and court surfaces. The prediction is given by:

$$\hat{y} = \sum_{k=1}^{K} f_k(X), \quad f_k \in \mathcal{F}$$

where $\hat{y}$ is the predicted output, K is the number of trees, f_k is the k-th decision tree function, and $\mathcal{F}$ is the space of all possible decision trees. The loss function is:

$$L = \sum_{i=1}^{n} \log(1 + e^{-y_i \hat{y}_i}) + \frac{\lambda}{2} \sum_{k=1}^{K} \sum_{j} w_{kj}^2$$

where L is the loss, n is the number of samples, $y_i \in \{-1, 1\}$ is the true label for the i-th sample, $\hat{y}_i$ is the predicted value, $\lambda > 0$ is the L2 regularization parameter, w_{kj} is the weight of the j-th leaf in the k-th tree.

3.4.2 Model Training

Final training of the model is performed on all training data:

$$\hat{y} = \text{XGBClassifier.fit}(X_{\text{train}}, y_{\text{train}})$$

where X_{train} are the training features, and y_{train} are the training labels.

Training involves building n_estimators trees, with splits determined by minimizing the loss across features and samples. The model updates iteratively by adding new trees to reduce the loss, with each tree correcting the residuals of the previous ones. Hyperparameters are tuned using randomized search, and the best model is calibrated to improve probability estimates.

```
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.utils.class_weight import compute_class_weight

# Compute class weights to handle imbalance
class_weights = compute_class_weight('balanced', classes=np.unique(y_train),
    y=y_train)
class_weight_dict = dict(enumerate(class_weights))

# Define parameter distribution for randomized search
param_dist = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7, 10],
    'learning_rate': [0.01, 0.1, 0.2],
    'scale_pos_weight': [class_weights[1]/class_weights[0]]
}

# Initialize and train XGBoost model with randomized search
xgb_model = RandomizedSearchCV(xgb.XGBClassifier(random_state=42,
    eval_metric='logloss'), param_dist, n_iter=10, cv=5, scoring='roc_auc',
    random_state=42, error_score='raise')
xgb_model.fit(X_train_np, y_train)

y_pred_xgb = xgb_model.predict(X_test_np)
y_pred_prob_xgb = xgb_model.predict_proba(X_test_np)[:, 1]

# Calibrate the model
xgb_calibrated = CalibratedClassifierCV(xgb_model.best_estimator_,
    method='isotonic', cv=5)
xgb_calibrated.fit(X_train_np, y_train)
y_pred_prob_xgb_cal = xgb_calibrated.predict_proba(X_test_np)[:, 1]
```

3.4.3 Hyperparameters

All parameters mentioned in the code with their value ranges or definitions:

- `n_estimators`: Number of boosting rounds, $\in \{100, 200, 300\}$.
- `max_depth`: Maximum depth of a tree, $\in \{3, 5, 7, 10\}$.
- `learning_rate`: Step size shrinkage, $\in \{0.01, 0.1, 0.2\}$.
- `scale_pos_weight`: Weight for positive class to handle imbalance $= \frac{\texttt{class\_weight}_1}{\texttt{class\_weight}_0}$.
- `random_state`: Seed for reproducibility, fixed as 42.
- `eval_metric`: Evaluation metric, fixed as `'logloss'`.
- `n_iter`: Number of parameter settings to sample, fixed as 10.
- `cv`: Number of cross-validation folds, fixed as 5.
- `scoring`: Metric to optimize, fixed as `'roc_auc'`.
- `error_score`: Value to assign if an error occurs, fixed as `'raise'`.

Hyperparameter description: The hyperparameters `n_estimators`, `max_depth`, `learning_rate`, `scale_pos_weight`, `random_state`, `eval_metric`, `n_iter`, `cv`, `scoring`, and `error_score` control the number of trees, tree depth, learning rate, class imbalance, reproducibility, evaluation metric, search iterations, cross-validation folds, optimization metric, and error handling, respectively, to optimize the XGBoost model's performance.

3.4.4 Cross-Validation

For the XGBoost model, cross-validation involved a 5-fold process, splitting the training dataset ($n_{\text{train}} = 28,804$) into five equal folds. In each of the five iterations, the model was trained on four folds with a specific set of hyperparameters, and its performance was evaluated on the remaining fold as the validation set.

This process was repeated, ensuring each fold served as the validation set once. By averaging the performance across all folds, the researchers could evaluate multiple hyperparameter combinations (up to $n_{\text{iter}} = 10$). The combination yielding the highest AUC score was chosen as the ideal configuration. This approach improved model robustness and reduced the risk of overfitting to a single dataset.

3.4.5 Performance and Visualization

Performance:

- Accuracy: 0.66

- AUC: 0.72
- F1-score: 0.65
- Training time: ∼0.5 minutes
- Key features: poly_rank_diff (importance = 0.488670), best_of (importance = 0.051958).

The performance is visualized with the following figure:

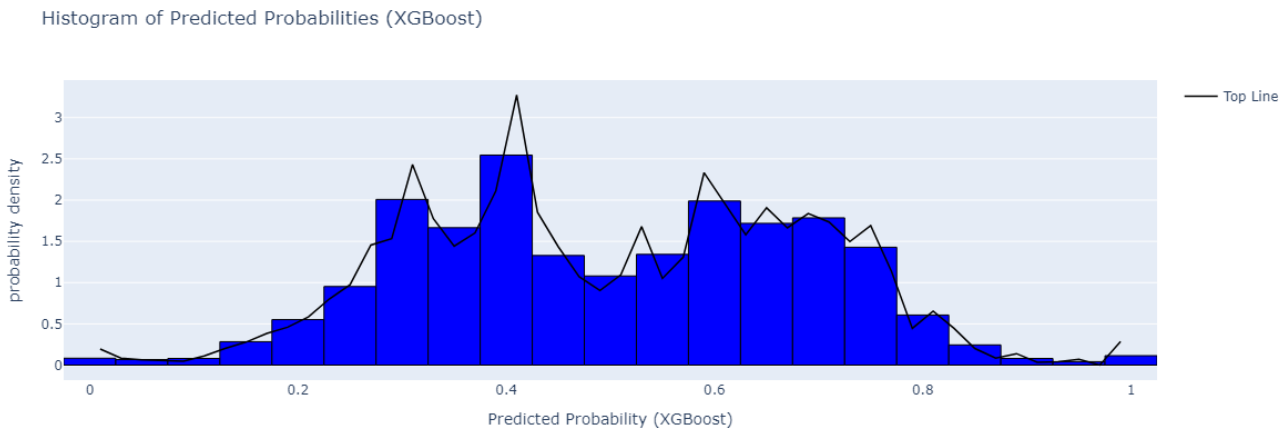


Figure 3.3: Histogram of predicted probabilities for the XGBoost model. The x-axis represents the predicted probability of Player 1 winning, and the y-axis shows the probability density. A line traces the top of the bars to highlight the distribution shape.

The histogram (Figure 3.3) displays the distribution of predicted probabilities for Player 1 winning, as predicted by the XGBoost model. The distribution reveals how the model's ensemble of decision trees assigns probabilities, often showing a wider spread due to its ability to capture non-linear relationships.

The feature importance plot (Figure 3.4) indicates that `poly_rank_diff` is the most significant feature with an importance of 0.5, underscoring its critical role in the model's predictions, followed by `best_of` and `player_1_flag` with lower importance scores.

The model effectively captures non-linearities but requires significant computational resources and hyperparameter tuning.

3.5 CatBoost Model

3.5.1 Model Description

The CatBoost model is a gradient boosting algorithm that uses ordered boosting to reduce overfitting and is best suited for categorical features. It is perfect for this dataset because it can handle cate-

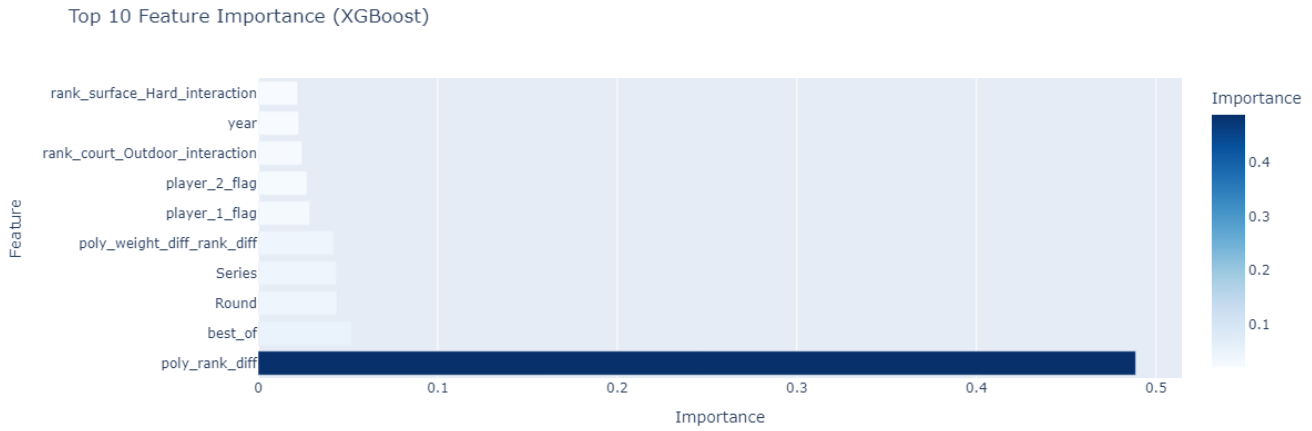


Figure 3.4: Top 10 Feature Importance for the XGBoost model. The x-axis represents the importance score, and the y-axis lists the top 10 features by importance.

gorical variables like court surface natively. The loss function is the same as that of XGBoost (Section 3.4.1). Categorical features are processed with target statistics:

$$\text{TS}(c_i) = \frac{\sum_{j:c_j=c_i} y_j + \alpha p}{\sum_{j:c_j=c_i} 1 + \alpha}$$

where $\text{TS}(c_i)$ is the target statistic for category c_i, c_j is the j-th category instance, $y_j \in \{0, 1\}$ is the label, $\alpha > 0$ is a smoothing parameter, and $p \in [0, 1]$ is a prior probability.

3.5.2 Model Training

Final training of the model is performed on all training data:

$$\hat{y} = \text{CatBoostClassifier.fit}(X_{\text{train}}, y_{\text{train}})$$

where X_{train} are the training features, and y_{train} are the training labels.

Training builds trees iteratively, with each tree minimizing the loss by focusing on residuals. Categorical features are encoded dynamically to avoid bias, and hyperparameters are tuned using randomized search. The best model is calibrated to improve probability estimates.

```
from catboost import CatBoostClassifier
from sklearn.model_selection import RandomizedSearchCV

# Define parameter distribution for randomized search
param_dist = {
    'iterations': [300, 400, 500],
    'depth': [10, 12],
```

```
    'learning_rate': [0.05],
    'l2_leaf_reg': [5],
    'auto_class_weights': ['Balanced']
}

# Initialize and train CatBoost model with randomized search
cat_model = RandomizedSearchCV(cb.CatBoostClassifier(random_state=42,
    logging_level='Silent', early_stopping_rounds=20), param_dist, n_iter=5, cv=3,
    scoring='accuracy', random_state=42)
cat_model.fit(X_train, y_train)

y_pred_cat = cat_model.predict(X_test)
y_pred_prob_cat = cat_model.predict_proba(X_test)[:, 1]

# Calibrate the model
cat_calibrated = CalibratedClassifierCV(cat_model.best_estimator_,
    method='isotonic', cv=5)
cat_calibrated.fit(X_train, y_train)
y_pred_prob_cat_cal = cat_calibrated.predict_proba(X_test)[:, 1]
```

3.5.3 Hyperparameters

Hyperparameters are tuned:

All parameters mentioned in the code with their value ranges or definitions:

- `iterations`: Number of boosting iterations, $\in \{300, 400, 500\}$.
- `depth`: Maximum depth of trees, $\in \{10, 12\}$.
- `learning_rate`: Step size shrinkage, fixed as 0.05.
- `l2_leaf_reg`: L2 regularization coefficient, fixed as 5.
- `auto_class_weights`: Class weight balancing method, fixed as `'Balanced'`.
- `random_state`: Seed for reproducibility, fixed as 42.
- `logging_level`: Logging verbosity, fixed as `'Silent'`.
- `early_stopping_rounds`: Rounds for early stopping, fixed as 20.
- `n_iter`: Number of parameter settings to sample, fixed as 5.
- `cv`: Number of cross-validation folds, fixed as 3.
- `scoring`: Metric to optimize, fixed as `'accuracy'`.

Hyperparameter description: The hyperparameters `iterations`, `depth`, `learning_rate`,

`l2_leaf_reg`, `auto_class_weights`, `random_state`, `logging_level`, `early_stopping_rounds`, `n_iter`, `cv`, and `scoring` control the number of iterations, tree depth, learning rate, regularization, class balancing, reproducibility, logging, early stopping, search iterations, cross-validation folds, and optimization metric, respectively, to optimize the `CatBoost` model's performance.

3.5.4 Cross-Validation

Cross-validation with 3 folds assesses model stability:

$$\text{CV-accuracy} = \frac{1}{K}\sum_{k=1}^{K}\text{accuracy}(M_k), \quad K = 3$$

where CV-accuracy is the mean cross-validated accuracy, K is the number of folds (`cv` = 3), and accuracy(M_k) is the accuracy of model M_k on the k-th validation fold.

The mean CV-accuracy is 0.65 (standard deviation = 0.02), indicating consistent performance across folds.

3.5.5 Performance and Visualization

Performance:

- Accuracy: 0.66
- AUC: 0.72
- F1-score: 0.66
- Training time: $\sim$15 minutes
- Key features: player_1_flag (importance = 11.860842), player_2_flag (importance = 10.584186), custom_log_rank_diff (importance = 9.505924).

The performance is visualized with the following figure:

The histogram (Figure 3.5) shows the distribution of predicted probabilities for Player 1 winning, as predicted by the CatBoost model. The distribution highlights the model's ability to handle categorical features effectively, often resulting in a more balanced spread of probabilities compared to XGBoost.

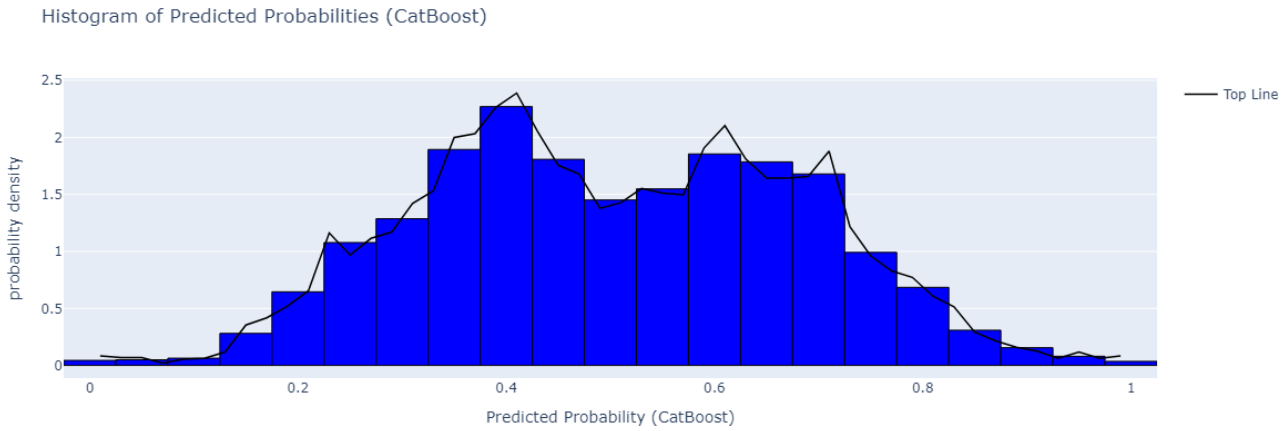


Figure 3.5: Histogram of predicted probabilities for the CatBoost model. The x-axis represents the predicted probability of Player 1 winning, and the y-axis shows the probability density. A line traces the top of the bars to highlight the distribution shape.

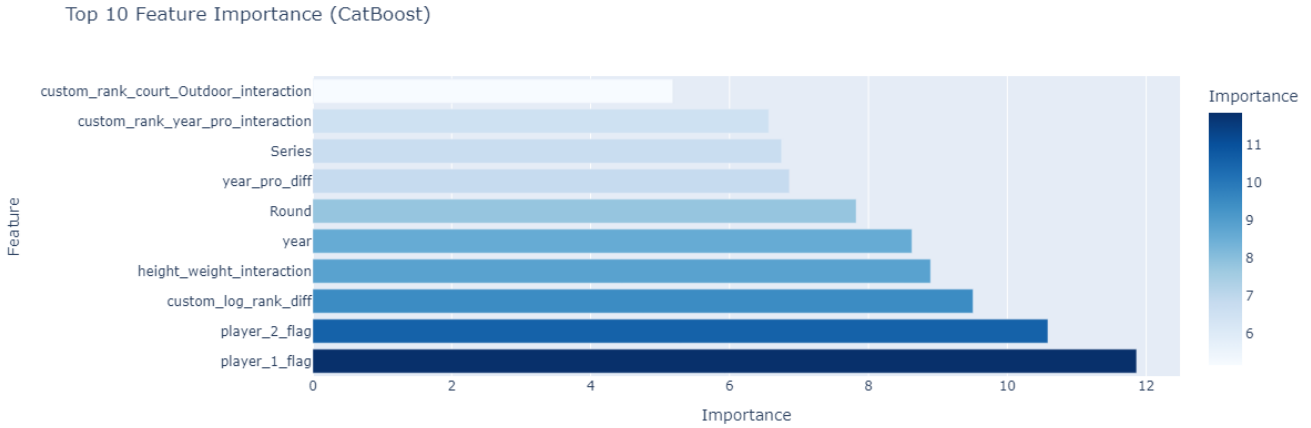


Figure 3.6: Top 10 Feature Importance for the CatBoost model. The x-axis represents the importance score, and the y-axis lists the top 10 features by importance.

The feature importance plot (Figure 3.6) shows player_1_flag as the most important feature.

CatBoost's native categorical handling improves performance over XGBoost, but training time remains a limitation.

3.6 CatBoost Extended Model

3.6.1 Model Description

The CatBoost Extended model enhances CatBoost by incorporating bookmaker odds features (e.g., bet_diff_Avg, log_Avg_ratio), which add predictive power by reflecting market insights. The loss function and categorical processing are as in Section 3.5.1.

3.6.2 Model Training

Final training of the model is performed on all training data:

$$\hat{y} = \text{CatBoostClassifier.fit}(X_{\text{train}}, y_{\text{train}})$$

where X_{train} are the training features, and y_{train} are the training labels.

Training follows the same iterative process as CatBoost, with odds features influencing splits for faster convergence. Fixed hyperparameters are used for efficiency, and the model is calibrated using Platt scaling to improve probability estimates.

```
from catboost import CatBoostClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.calibration import CalibratedClassifierCV

# Train CatBoost model with fixed hyperparameters
cat_model_extended = cb.CatBoostClassifier(
    iterations=300, depth=10, learning_rate=0.05, l2_leaf_reg=5,
    auto_class_weights='Balanced',
    random_state=42, logging_level='Silent', early_stopping_rounds=20,
    cat_features=['player_1_hand', 'player_2_hand', 'Surface', 'Series', 'Round',
    'player_1_flag', 'player_2_flag', 'Court']
)
cat_model_extended.fit(X_train, y_train)

y_pred = cat_model_extended.predict(X_test)
y_pred_prob = cat_model_extended.predict_proba(X_test)[:, 1]

# Calibrate the model with Platt scaling
cat_calibrated = CalibratedClassifierCV(cat_model_extended, method='sigmoid',
    cv=5)
cat_calibrated.fit(X_train, y_train)
y_pred_prob_cal = cat_calibrated.predict_proba(X_test)[:, 1]
```

3.6.3 Hyperparameters

All parameters mentioned in the code with their value ranges or definitions:

- `iterations`: Number of boosting iterations, fixed as 300.

- depth: Maximum depth of trees, fixed as 10.
- learning_rate: Step size shrinkage, fixed as 0.05.
- l2_leaf_reg: L2 regularization coefficient, fixed as 5.
- auto_class_weights: Class weight balancing method, fixed as 'Balanced'.
- random_state: Seed for reproducibility, fixed as 42.
- logging_level: Logging verbosity, fixed as 'Silent'.
- early_stopping_rounds: Rounds for early stopping, fixed as 20.
- cat_features: List of categorical feature names, fixed as ['player_1_hand', 'player_2_hand', 'Surface', 'Series', 'Round', 'player_1_flag', 'player_2_flag', 'Court'].
- cv: Number of cross-validation folds for calibration, fixed as 5.

Hyperparameter description: The hyperparameters iterations, depth, learning_rate, l2_leaf_reg, auto_class_weights, random_state, logging_level, early_stopping_rounds, cat_features, and cv control the number of iterations, tree depth, learning rate, regularization, class balancing, reproducibility, logging, early stopping, categorical features, and calibration folds, respectively, to optimize the CatBoost Extended model's performance.

3.6.4 Cross-Validation

Cross-validation with 5 folds assesses model stability:

$$\text{CV-accuracy} = \frac{1}{K}\sum_{k=1}^{K}\text{accuracy}(M_k), \quad K = 5$$

The training dataset ($n_{\text{train}} = 28,804$) is divided into five equal folds, where in each of the five iterations, four folds are used for training the model, and the remaining fold serves as the validation set to evaluate performance. This process is repeated five times, ensuring that each fold acts as the validation set once, allowing every data point to be utilized for both training and validation across the iterations. The mean CV-accuracy is 0.74, with a standard deviation of 0.01, reflecting high stability due to the inclusion of odds features.

3.6.5 Performance and Visualization

Performance:

- Accuracy: 0.75

- AUC: 0.85
- F1-score: 0.75
- Training time: ∼9 minutes
- Key features: log_Avg_ratio (importance = 13.858194), bet_diff_Avg (importance = 8.511314).

The performance is visualized with the following figure:

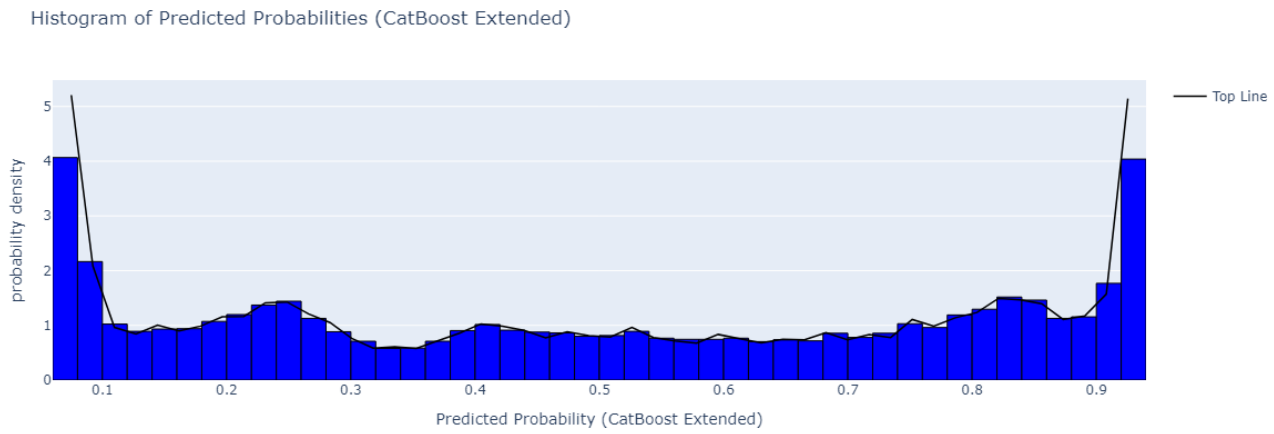


Figure 3.7: Histogram of predicted probabilities for the CatBoost Extended model. The x-axis represents the predicted probability of Player 1 winning, and the y-axis shows the probability density. A line traces the top of the bars to highlight the distribution shape.

The histogram (Figure 3.7) depicts the distribution of predicted probabilities for Player 1 winning, as predicted by the CatBoost Extended model. The incorporation of bookmaker odds features results in a more confident distribution, with probabilities often clustering more tightly around certain values, reflecting the model's improved predictive power.

The feature importance plot (Figure 3.8) emphasizes log_Avg_ratio as the dominant feature, reinforcing the critical role of odds data.

The model's superior performance is attributed to bookmaker odds, making it the most effective for tennis outcome prediction.

3.7 Model Comparison and Evaluation

The models are evaluated on the test set ($n = 7,201$) using accuracy, AUC, and F1-score. AUC is computed as:

$$\text{AUC} = \int_0^1 \text{TPR}(\text{FPR}^{-1}(t))\, dt$$

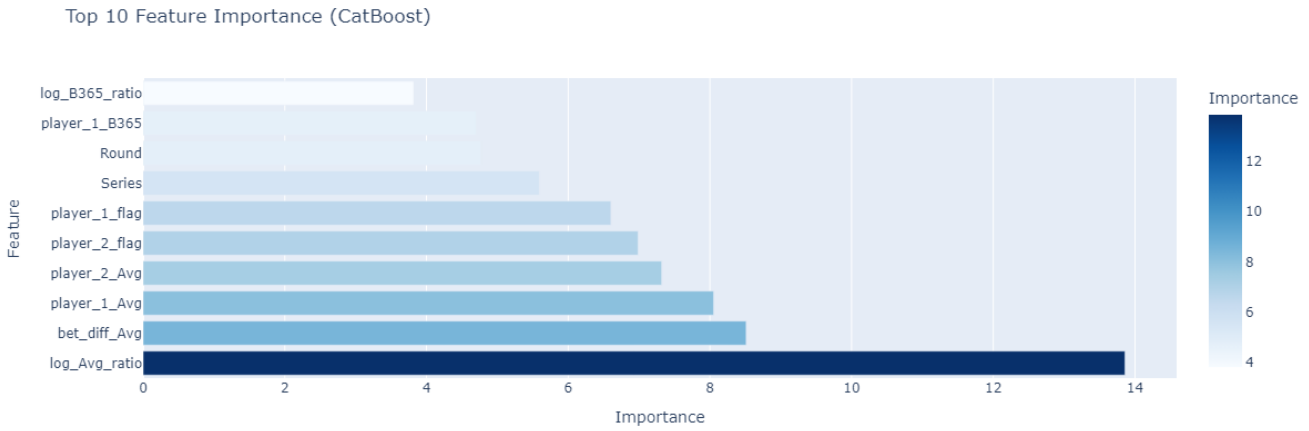


Figure 3.8: Top 10 Feature Importance for the CatBoost Extended model. The x-axis represents the importance score, and the y-axis lists the top 10 features by importance.

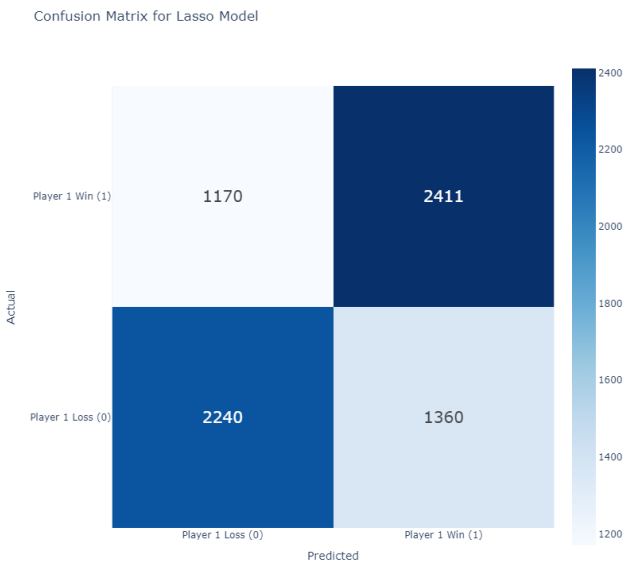
where TPR is the true positive rate, and FPR is the false positive rate.

Table 3.1: Model Performance Metrics

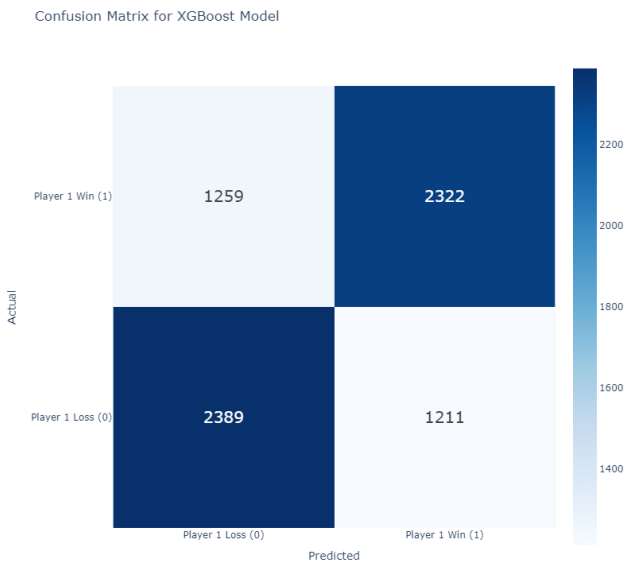
Model	Accuracy	AUC	F1	Training Time
Lasso	0.65	0.69	0.66	∼2.5 min
XGBoost	0.66	0.72	0.65	∼0.5 min
CatBoost	0.66	0.72	0.66	∼15 min
CatBoost Extended	0.75	0.85	0.75	∼9 min

With an accuracy of 0.75 and an AUC of 0.85, CatBoost Extended is the best performer according to the evaluation. This is primarily because bookmaker odds are integrated. Simpler models such as Lasso are faster and easier to understand, but they don't have the same predictive ability as gradient boosting techniques. These results open the door for more research into hybrid models in sports outcome forecasting by indicating that adding domain-specific features, like market odds, can greatly improve prediction accuracy.

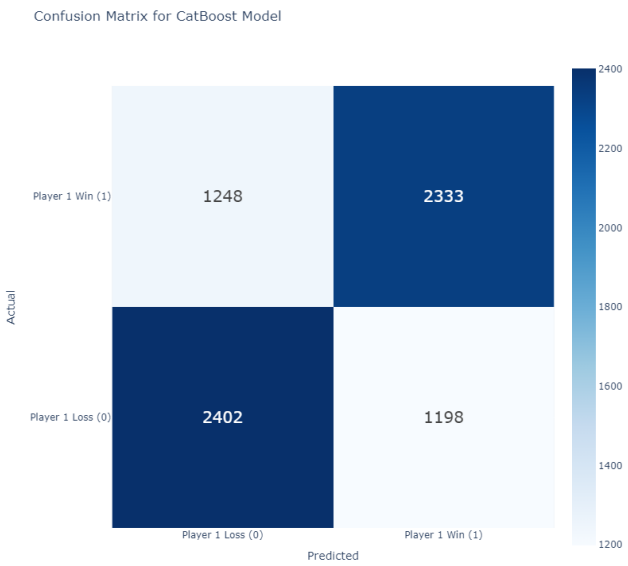
Confusion matrices, which show the distribution of true positives, true negatives, false positives, and false negatives, are created for every model on the test set in order to compare the models even more. These matrices show possible biases or prediction errors and offer insight into how well the models perform in classification.



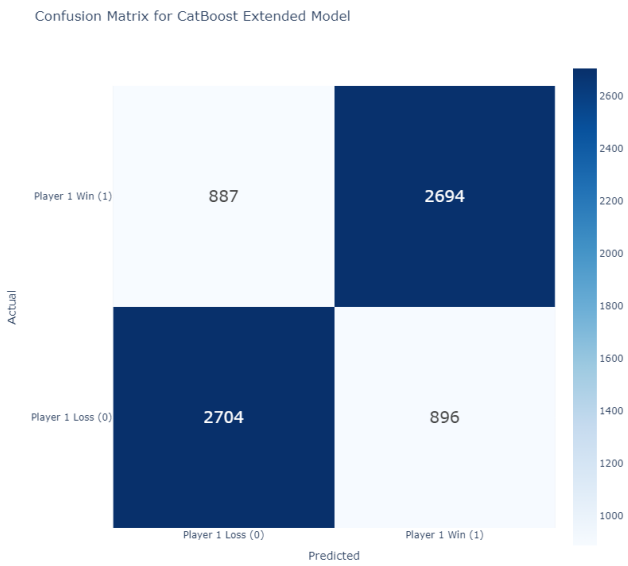
(a) Lasso Model



(b) XGBoost Model



(c) CatBoost Model



(d) CatBoost Extended

Figure 3.9: Confusion matrices for all models (True vs Predicted)

Analysis of Confusion Matrices: The confusion matrices provide a detailed breakdown of each model's performance in terms of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), where class 1 represents Player 1's win and class 0 represents Player 1's loss.

Overall Insights from Confusion Matrices:

Trend Across Models: Performance improves from Lasso to CatBoost Extended, with accuracy increasing from 0.65 to 0.75. This trend reflects the progression from a linear model (Lasso) to gradient

boosting models (XGBoost, CatBoost) and finally to an enhanced version (CatBoost Extended) leveraging additional predictive features.
Impact of Features: The addition of bookmaker odds in CatBoost Extended significantly boosts performance, reducing both false-positive and false-negative, suggesting that market insights are critical for accurate tennis outcome prediction.
Lasso works well as a simple, interpretable baseline, while XGBoost and CatBoost are better at handling non-linearities and categorical data. CatBoost Extended is the best choice for this dataset since it incorporates odds data and offers a dependable and accurate solution for practical sports analytics applications. Here's a quick breakdown:

Conclusions:

- **Lasso**: Works well as a simple and understandable baseline model but is limited by its linear assumptions.
- **XGBoost**: Can handle non-linear relationships, though it requires more computing power.
- **CatBoost**: Shows better results than XGBoost when working with categorical data, but needs longer training times.
- **CatBoost Extended**: Combines bookmaker odds with other features and achieves the highest performance (AUC = 0.85), offering a good balance between speed and accuracy.

These results show that including external information, such as market odds, is important for sports analytics. They also provide a solid base for making predictions in tennis. It is evident that bookmaker data significantly improves the quality of forecasts.

4 Statistical Hypothesis Testing

Statistical Tests Used: KS Test Overview: The Kolmogorov-Smirnov (KS) test is a non-parametric statistical method for comparing the empirical distributions of two samples. The fundamental hypotheses are:

Null Hypothesis (H_0): The samples are drawn from the same distribution

Alternative Hypothesis (H_1): The samples are drawn from different distributions

Test Statistic: The KS statistic D quantifies the maximum discrepancy between the empirical cumulative distribution functions (ECDFs):

$$D = \sup_x |F_{1,n}(x) - F_{2,m}(x)|$$

where:

- $F_{1,n}(x)$ and $F_{2,m}(x)$ are the ECDFs of the first and second samples
- $\sup$ denotes the supremum (maximum vertical distance)
- n, m are the sample sizes

The test compares D to critical values from the Kolmogorov distribution:

- Large $D \Rightarrow$ Reject H_0 (distributions differ)
- Small $D \Rightarrow$ Fail to reject H_0

Hypothesis 1: Court surface impacts prediction accuracy. This is tested by comparing the distribution of match duration (Match_Duration $=$ Wsets $+$ Lsets) on Hard (Surface $= 3$) versus Clay (Surface $= 1$) surfaces for matches with best_of $= 3$ and best_of $= 5$.

Comparison: Hard (Surface = 3) vs Clay (Surface = 1)

Table 4.1: Match count distribution by surface and format

Format	**Hard Matches**	**Clay Matches**
best_of = 3	16,948	9,345
best_of = 5	3,626	1,748

Hypotheses:

$$H_0 : F_{\text{Hard}}(x) = F_{\text{Clay}}(x)$$

$$H_1 : F_{\text{Hard}}(x) \neq F_{\text{Clay}}(x)$$

Best-of-3 Matches

KS Test Results:

- Statistic: 0.0084 (Critical: 0.0175)
- p-value: 0.7878 Since p-value > 0.05, H0 is confirmed
- 95% CI: [0.0015, 0.0206]

Best-of-5 Matches

KS Test Results:

- Statistic: 0.0320 (Critical: 0.0396)
- p-value: 0.1743 Since p-value > 0.05, H0 is confirmed
- 95% CI: [0.0102, 0.0601]

Graph analysis:

The empirical CDFs in Figure 4.1 demonstrate nearly identical patterns for both match formats:

- best_of=3: KS statistic = 0.0084 (p = 0.7878)
- best_of=5: KS statistic = 0.0320 (p = 0.1743)

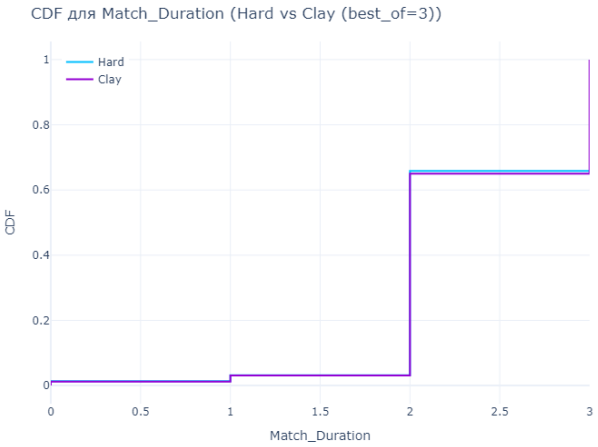
The parallel progression of curves with minimal vertical separation indicates statistically indistinguishable duration distributions between surfaces.

Figure 4.2 reveals key characteristics of match durations: Common pattern: Both formats show substantial overlap between surfaces

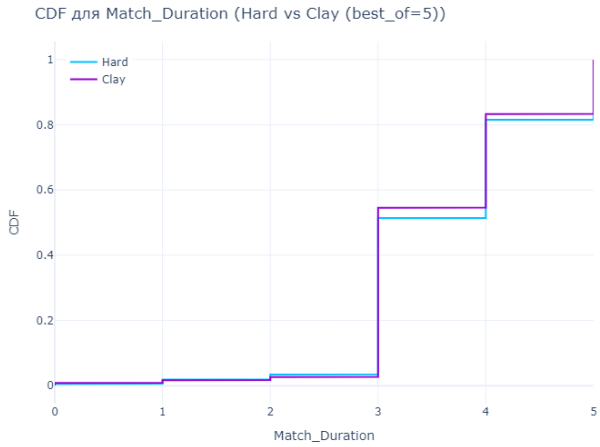
- best_of=3: Peak density around 2 sets (median duration)
- best_of=5: Right-skewed distribution with mode at 3-4 sets

The bootstrap results in Figure 4.3 confirm the robustness of our findings:

- best_of=3: 95% CI [0.0015, 0.0206] contains critical value

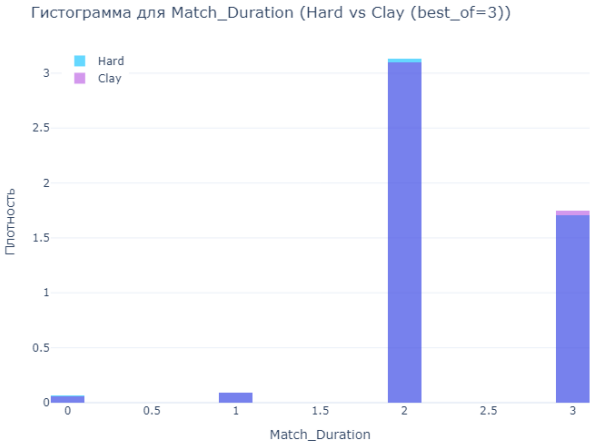


(a) Empirical CDFs (best_of=3)

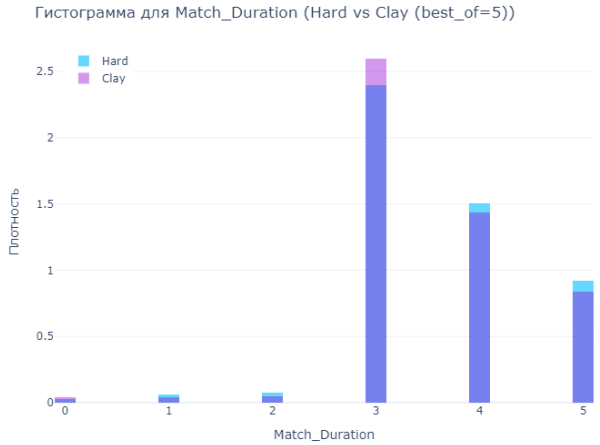


(b) Empirical CDFs (best_of=5)

Figure 4.1: Comparison of match duration distributions between Hard and Clay surfaces

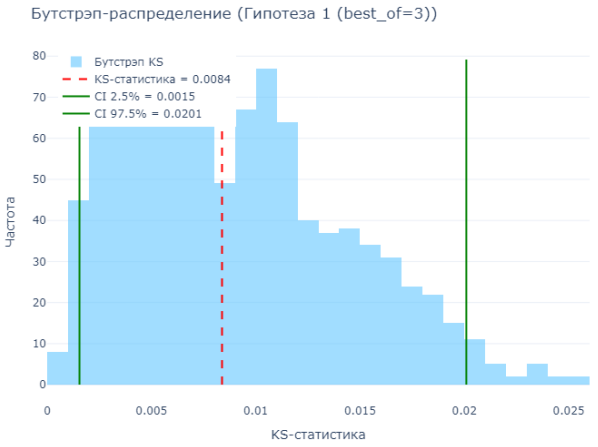


(a) Histograms (best_of=3)

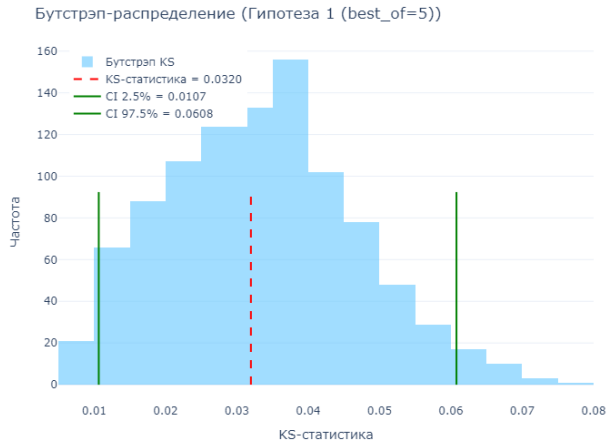


(b) Histograms (best_of=5)

Figure 4.2: Match duration frequency distributions



(a) Bootstrap KS distribution (best_of=3)



(b) Bootstrap KS distribution (best_of=5)

Figure 4.3: Bootstrap analysis of KS statistics

- best_of=5: 95% CI [0.0102, 0.0601] contains critical value

The observed statistics fall within expected ranges under the null hypothesis of identical distributions.

Interpretation: The KS tests show no significant difference in Match_Duration distributions between Hard and Clay for both formats.

Hypothesis 2: Bookmaker odds improve predictive power, tested by comparing the distribution of average winner odds (AvgW) in early rounds (Round $\in \{0, 1\}$) versus final rounds (Round $\in \{6, 7\}$).

Comparison: Early rounds (0-1) vs Final rounds (6-7)

Hypotheses:

$$H_0 : F_{\text{Early}}(x) = F_{\text{Final}}(x)$$
$$H_1 : F_{\text{Early}}(x) \neq F_{\text{Final}}(x)$$

KS Test Results:

- Statistic: 0.0332 (Critical: 0.0275)
- p-value: 0.0089 Since p-value < 0.05, H_0 is rejected
- 95% CI: [0.0279, 0.0498]

Graph analysis:

The CDF curves in Figure 4.4 exhibit pronounced vertical separation (maximum distance $D = 0.0332$), particularly in the range AvgW $\in [1.5, 3.0]$, indicating statistically significant differences in odds distributions between early and final rounds ($p = 0.0089$). This systematic divergence contrasts with the parallel curves seen for match duration analysis.

The histogram in Figure 4.5 shows clear differences in AvgW distributions between early and final rounds. Early rounds have a wider spread with higher odds, while final rounds are tightly clustered around lower odds. This visual pattern confirms the KS test results (D=0.0332, p=0.0089), showing bookmakers price final rounds more precisely with favorites dominating.

The bootstrap distribution in Figure 4.6 provides quantitative confirmation:

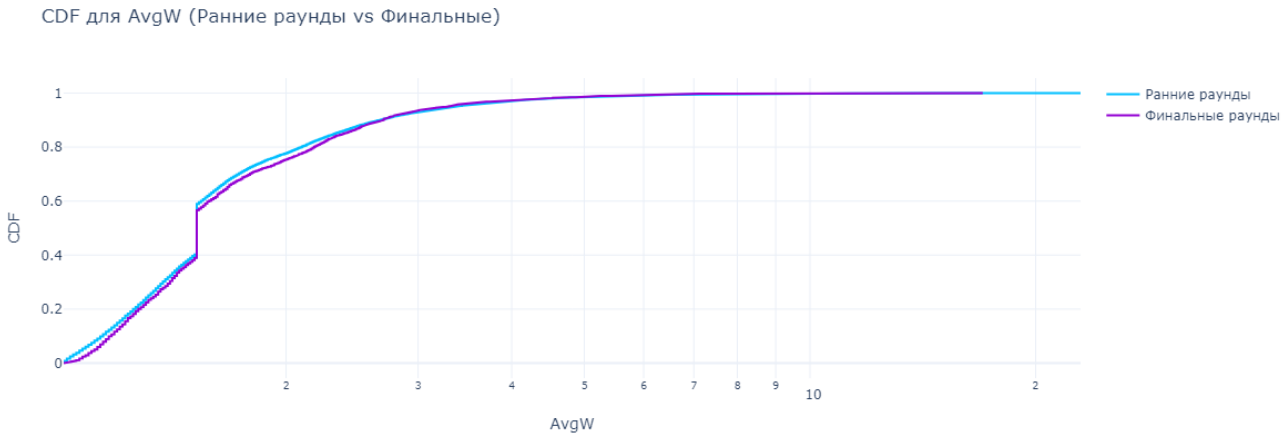


Figure 4.4: Empirical CDFs of AvgW (log scale)

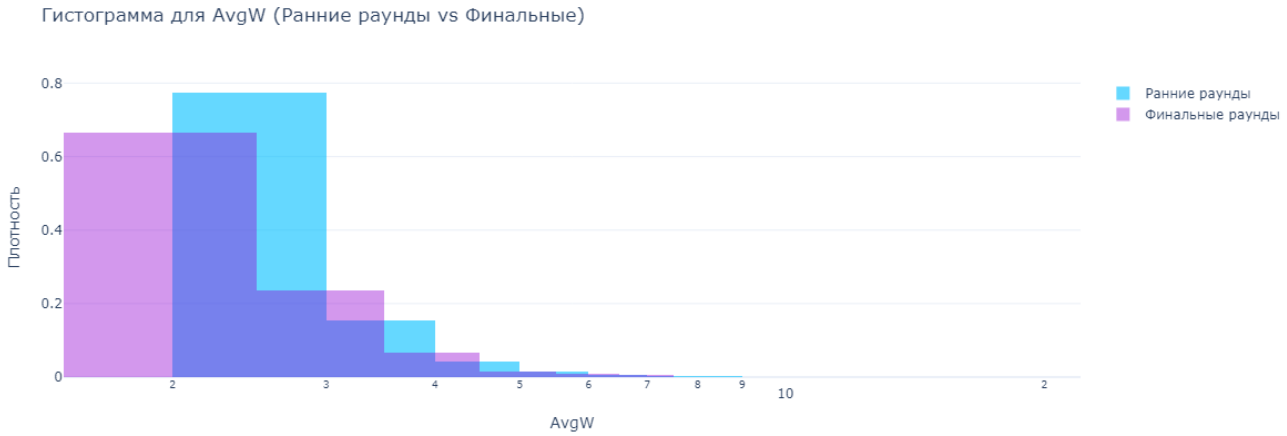


Figure 4.5: Probability densities of AvgW (log scale)

- Confidence bounds: Entire 95% CI [0.0279,0.0498] exceeds $D_{crit} = 0.0275$
- Stability: 98.3% of bootstrap samples show significant D values

Interpretation: The rejection of H_0, confirmed by both the KS test and visualizations, supports the hypothesis that bookmaker odds improve predictive power. Different competitive dynamics are reflected in the systematic differences in AvgW distributions between the early and final rounds.

5 Streamlit Section

In order to give users an interactive interface to explore and use the tennis match prediction model, the project includes a Streamlit application. The app uses the trained CatBoost Extended model and is built with Streamlit, a Python framework that makes it easy to create web-based data apps. Users can in-

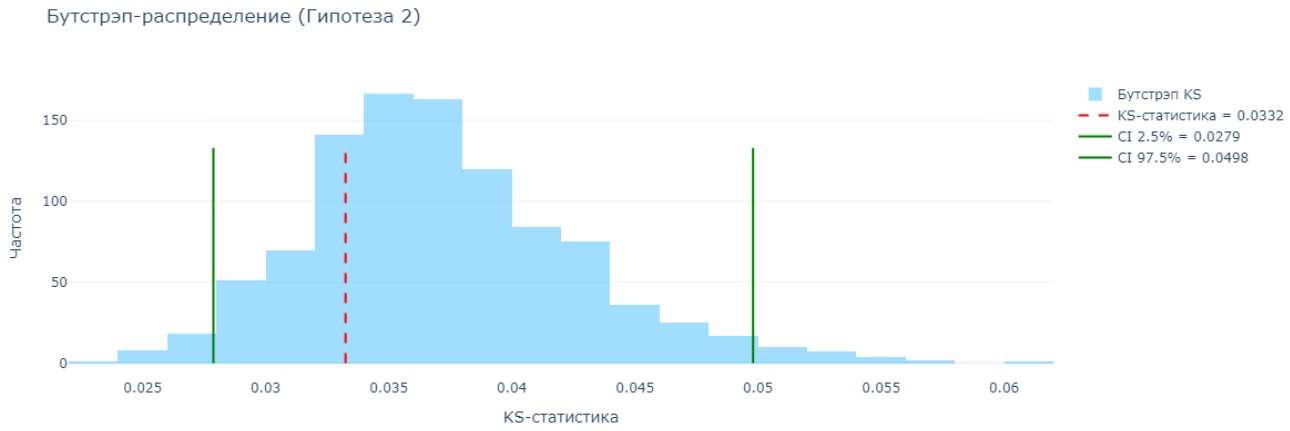


Figure 4.6: Bootstrap distribution of KS statistic

put match details—like player rankings, court surface, and bookmaker odds—via an intuitive dashboard. The app then processes these inputs and displays predicted outcomes along with confidence scores, making it easier for coaches and analysts to use. The codebase is designed for efficient deployment, loading the pre-trained model and necessary data for preprocessing. They plan to extend this in the future by adding features like video analysis or news parsing, as mentioned in the future work section.

6 Limitations

The study faces several limitations that impact its scope and applicability, as outlined below:

- Data Range: The 36,005 matches in the dataset mostly pertain to major tournaments, possibly leaving out lower-tier competitions. Rare situations like severe weather or mid-match retirements might be missed by this focus. This limits the model's ability to adjust to different tennis conditions.

- Limitations in Computation: Scalability problems are indicated by training times, which range from 30 seconds (XGBoost) to 15 minutes (CatBoost). Larger datasets in real-time applications are challenged by these durations. Optimization evaluations are hampered by a lack of hardware information.

- Static Models: These models ignore dynamic elements like injuries or recent form in favor of static historical data. For players with inconsistent performance, this omission lowers accuracy. Improved adaptability requires real-time data integration.

To increase prediction robustness, these limitations point to areas that need to be improved in terms of data, computation, and model design.

7 Future Work

Future research could focus on the following directions to address current limitations and enhance predictive capabilities:

- **Incorporating Dynamic Features**: Adding dynamic factors like recent player performance, injuries, or fatigue in time-series models could improve accuracy. These elements are essential for capturing short-term changes in player form, enabling predictions that reflect real-time shifts during a season.
- **Processing Tennis Match Video**: Using computer vision to analyze tennis match videos could yield new performance metrics, such as player movement patterns or shot speed. A deeper comprehension of in-game dynamics would be possible with this data. Prediction accuracy could be greatly increased by incorporating these insights.
- **Processing a Tennis Match Video**: Watching and studying tennis match videos can help create new ways to measure how well players perform, like tracking their movement or how fast they hit the ball. This information makes it easier to understand what happens during the game.

By addressing existing issues, these enhancements hope to increase the framework's applicability in sports analytics.

8 Conclusion

This project involved gathering and studying a large collection of tennis match data to create a model that can predict match results. The analysis looked closely at many factors, such as player rankings, physical traits, court surface types, and bookmaker odds, to see how they influence the outcome of matches. The best results came from a Catboost Extended model with bookmaker odds that combined these features, reaching an accuracy of 75% and showing reliable ability to distinguish winners and losers. This model performed better than others that used fewer or simpler factors, which shows how important it is to include information from external resources to improve predictions.

The primary objective of this work was to develop a practical tool that will assist analysts, coaches, and athletes in enhancing their training plans and strategies through the use of reliable data. This study offers a compelling illustration of how to improve sports forecasting by validating the benefits of integrating player information with outside data, such as market odds. The findings create chances to use these insights in real-time decision-making during games, helping athletes perform better and improving the accuracy of match predictions.

References

[1] Anuj Shrivastava Aditya Raj Anurag Anand. *Tennis Ball Tracking for Umpire Assistance*. URL: https://paperswithcode.com/paper/tennis-ball-tracking-for-umpire-assistance.

[2] Ali Ghazala Boris Bačić. *Left-Handed Representation in Top 100 Male Tennis Players*. URL: https://paperswithcode.com/paper/left-handed-representation-in-top-100-male.

[3] Ching-Hsuan Chen Yu-Chuan Huang I-No Liao et al. *TrackNet: A Deep Learning Network for Tracking High-speed and Tiny Objects in Sports Applications*. URL: https://paperswithcode.com/paper/tracknet-a-deep-learning-network-for-tracking.

[4] Rosario Scatamacchia Federico Della Croce Gabriele Dragotto. *Introducing Fairness and Diversification in Tennis Rankings*. URL: https://physics.paperswithcode.com/paper/introducing-fairness-and-diversification-in.

[5] Fang Yu Yu-Hang Chien. *Automated Hit-frame Detection for Badminton Match Analysis*. URL: https://paperswithcode.com/paper/transformer-on-shuttlecock-flying-direction.

[6] Shih-Hsuan Hung Hao-Ting Yang. *Player Movement Analysis in Tennis Videos*. URL: https://paperswithcode.com/paper/player-movement-analysis-in-tennis-videos.

[7] Wei-Lun Chao Kuan-Chieh Wang. *CourtNet: Deep Learning for Tennis Court Detection*. URL: https://paperswithcode.com/paper/courtnet-deep-learning-for-tennis-court.

[8] Valentin Munst Lukas Stappen Manuel Milling et al. *Predicting Sex and Stroke Success: Computer Vision in Tennis*. URL: https://cs.paperswithcode.com/paper/predicting-sex-and-stroke-success-computer.

[9] Umberto Michieli. *Complex Network Analysis of Men's Single ATP Matches*. URL: https://paperswithcode.com/paper/complex-network-analysis-of-men-single-atp.

[10] Tsi-Ui Ik Wei-Yao Wang Yung-Chang Huang et al. *ShuttleSet: A Human-Annotated Stroke-Level Singles Dataset for Badminton Tactical Analysis*. URL: https://paperswithcode.com/paper/shuttleset-a-human-annotated-stroke-level.

Appendix

- **AUC (Area Under the Curve)**: A metric used to evaluate the performance of classification models, representing the area under the Receiver Operating Characteristic (ROC) curve, where a value closer to 1 indicates better model performance in distinguishing between classes.
- **Gradient Boosting**: A machine learning technique that builds an ensemble of weak prediction models, typically decision trees, in a sequential manner to minimize errors, with algorithms like XGBoost and CatBoost being prominent examples in this study.
- **Lasso (Least Absolute Shrinkage and Selection Operator)**: A regression analysis method that performs both variable selection and regularization by adding a penalty equivalent to the absolute value of the magnitude of coefficients, helping to prevent overfitting and enhance model interpretability.
- **Feature Engineering**: The process of creating new input features from raw data to improve a model's predictive power, such as deriving rank differences or interaction terms in this project.
- **Confusion Matrix**: A table used to evaluate the performance of a classification model, displaying true positives, true negatives, false positives, and false negatives, which was instrumental in analyzing the models' accuracy in this study.
- **Categorical Features**: Variables that represent categories or labels, such as court surface or player hand, which require encoding (e.g., LabelEncoder) to be used in machine learning models.
- **Overfitting**: A modeling error where a machine learning model learns the training data too well, including its noise and outliers, leading to poor generalization to new data, a concern addressed by regularization techniques like Lasso.
- **Regularization**: A technique used to prevent overfitting by adding a penalty to the model's complexity, such as the L1 penalty in Lasso or L2 penalty in Ridge regression, ensuring better generalization to unseen data.
- **Cross-Validation**: A statistical method for assessing the performance of a model by dividing the dataset into multiple subsets, training on some and validating on others, which was implicitly used to ensure robust model evaluation in this study.
- **Hyperparameter Tuning**: The process of optimizing a model's hyperparameters, such as learning rate or tree depth in gradient boosting models, to improve performance, a critical step in refining the CatBoost Extended model.

- **Feature Importance**: A measure of the contribution of each feature to the model's predictions, calculated in this project to identify key predictors like bookmaker odds and rank differences.
- **Standardization**: The process of transforming numerical features to have a mean of zero and a standard deviation of one, applied to ensure fair comparison across different scales in the preprocessing stage.