University Football Injury Prediction

Malak El-Hamshary (ID:91241072)
Rowida Mohammed (ID: 91240303)
Mona Mohammed Elkholy (ID: 91241075)
Khadija Zakaria Mabrouk (ID: 91240965)

I. Introduction

Football is a high-intensity sport with a notable risk of injury, especially among university athletes balancing competitive play with academics. Injuries affect both player availability and long-term health, emphasizing the need for effective prediction and prevention strategies. This study uses the University Football Injury Prediction Dataset [1], containing data from 800 Chinese university players, including physical characteristics, fitness metrics, workload, lifestyle factors, and training adherence. The target variable indicates whether a player sustained a medically verified injury in the following season.

In this work, machine learning classification techniques are applied, focusing on a binary injury prediction task using the Naïve Bayes (NB) classifier. A custom NB model is implemented from scratch, trained on processed training data, and evaluated on the testing set. Results are compared with standard Python implementations to assess the effectiveness and practical utility of the custom approach in predicting football injuries.

II. Methodology

A. Software Packages

All experiments were conducted using the Python programming language. The following software packages were utilized:

- `scikit-learn`: pre-implemented Gaussian Naive Bayes model and evaluation metrics
- `NumPy`: numerical operations
- `pandas`: data handling and preprocessing
- `Matplotlib` and `Seaborn`: data visualization
- `SciPy`: statistical tests (Shapiro-Wilk)
- `os`: interacting with the underlying operating system

B. Data Pre-Processing

1) Data Source and Description: The dataset used in this study was obtained from Kaggle's University Football Injury Prediction Dataset. It comprises biomechanical, physiological, and behavioral attributes of university football players. The target variable represents a binary classification indicating whether a player sustained an injury during the observation period. Key dataset characteristics include:

- **Samples:** 800 university football players
- **Features:** 18 input features + 1 target label
- **Task:** Binary classification (0 = No injury, 1 = Injury)
- **Balance:** Well-balanced class distribution
- **Age Range:** 18-24 years (typical university students)

Quantitative Features (16 variables): Age, Height, Weight, Training Hours Per Week, Matches Played Past Season, Previous Injury Count, Knee Strength Score, Hamstring Flexibility, Reaction Time ms, Balance Test Score, Sprint Speed 10ms, Agility Score, Sleep Hours Per Night, Stress Level Score, Nutrition Quality Score, and BMI.

Categorical Features (2 variables): Position and Warmup Routine Adherence.

Prior to model training, the categorical variable Position was transformed using Label Encoding to convert nominal categories into numerical representations suitable for machine learning algorithms.

2) Outlier Detection and Dataset Preparation: Outliers in quantitative features were identified using the Interquartile Range (IQR) method. For each feature, the first quartile $Q1$, third quartile $Q3$, and IQR ($Q3 - Q1$) were computed [2]. Data points falling outside the range:

$$[Q_1 - 2.0 \times \text{IQR}, \quad Q_3 + 2.0 \times \text{IQR}] \tag{1}$$

were flagged as outliers. Any observation containing at least one outlier across the quantitative attributes was removed from the dataset. This outlier removal procedure was applied to the complete dataset prior to train-test splitting, resulting in two parallel datasets for comparative analysis:

1) **Raw Dataset:** The original data with no outlier removal applied.
2) **Cleaned Dataset:** The data after IQR-based outlier removal.

A total of 24 observations were identified and removed during the cleaning process as shown in examples in Fig.1-2. Both datasets were visualized using box plots and histograms to assess distributional characteristics and the impact of outlier removal. This dual-dataset approach enabled a controlled comparison of model performance with and without the presence of outliers [3], providing insights into model robustness.

3) Descriptive Statistical Analysis: Comprehensive descriptive statistics were computed for both raw and cleaned datasets to characterize the data distribution and central tendencies.

Descriptive statistics were computed for both the raw and the cleaned datasets.

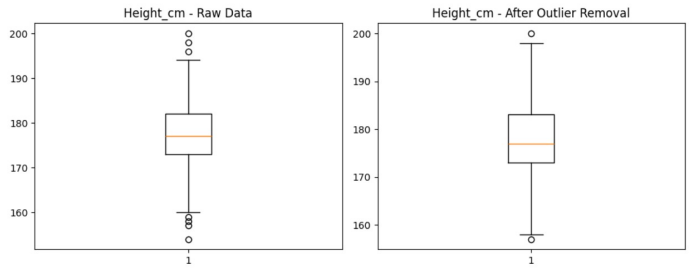

Fig. 1: Outlier reduction for Height feature.

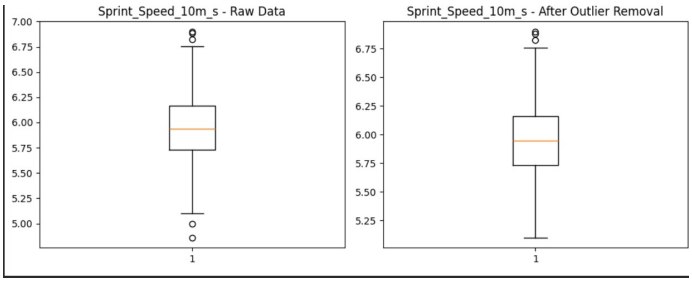

Fig. 2: Outlier removal for Sprint Speed feature.

a) For all quantitative features: Measures of central tendency (mean and median) and Measures of dispersion (variance, standard deviation, minimum, and maximum) were calculated.

b) For categorical variables: The mode was reported as the primary measure of central tendency.

All statistics were computed on unscaled data to maintain interpretability in the original measurement units and for cleaned and raw data.

4) Train-Test Data Partitioning: Both the raw and cleaned datasets were independently partitioned into training and testing subsets using an 80%-20% stratified split.

5) Feature Standardization: To ensure all quantitative features contributed equally to model training and to improve convergence in gradient-based algorithms, z-score standardization was applied. Critically, standardization parameters were computed exclusively from the training subset to prevent data leakage. For each quantitative feature, the transformation was defined as:

$$z = \frac{x - \mu_{\text{train}}}{\sigma_{\text{train}}} \tag{2}$$

where μ_{train} and σ_{train} represent the mean and standard deviation computed from the training data only.

This approach maintained the integrity of the train-test separation and provided a realistic evaluation of model generalization performance.

C. Visualization and Hypothesis Testing

This part of the project explored the statistical properties of the training data to validate assumptions for the Naive Bayes classifier. All analyses were conducted on the training set to prevent data leakage.

1) Distribution Analysis: Histograms of each quantitative feature were plotted in their original scale to preserve interpretability. Visual inspection of these plots, as shown in Figs. 3-6, allowed identification of key characteristics such as symmetry, central tendency, spread, and tails, and enabled qualitative classification of features as approximately Gaussian, skewed, or non-Gaussian.

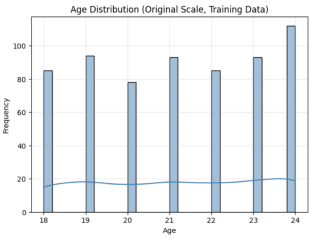

Fig. 3: Feature A distribution (with outliers).

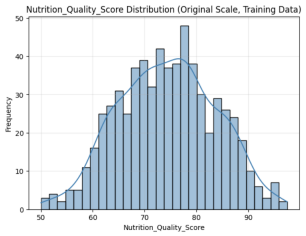

Fig. 4: Feature B distribution (with outliers).

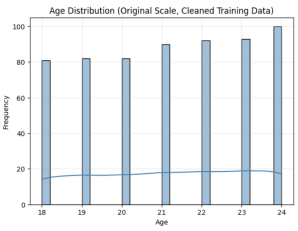

Fig. 5: Feature A distribution (outliers removed).

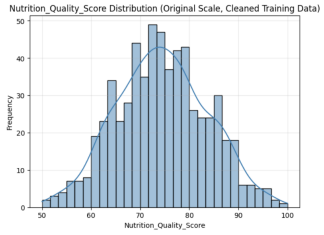

Fig. 6: Feature B distribution (outliers removed).

From the visual comparison, the *Age* shows an approximately uniform distribution in the raw data, which becomes even more uniform after outlier removal as extreme values are eliminated. *Nutrition Quality Score* initially has a broad Gaussian-like shape with a small secondary peak, but after outlier removal, it narrows and more closely resembles a single, well-defined Gaussian distribution.

2) Normality Testing: The Shapiro–Wilk test [4] was used to assess whether each quantitative feature follows a normal distribution, as it performs well for small to moderate sample sizes and continuous data. The hypotheses were defined as:

- Null Hypothesis (H_0): The feature is normally distributed.
- Alternative Hypothesis (H_1): The feature is not normally distributed.

The test was conducted at $\alpha = 0.05$. For each feature, the test statistic W and corresponding p-value were computed. Features with $p < \alpha$ led to rejection of H_0, indicating deviation from normality.

The Shapiro–Wilk statistic W measures how well the ordered sample values align with the expected order statistics of a normal distribution:

$$W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

where $x_{(i)}$ is the i-th sorted value, $\bar{x}$ is the sample mean, and a_i are constants based on normal order statistics. Values of W close to 1 indicate normality, while smaller values suggest deviation. The p-value quantifies the probability of observing such a deviation under normality, with small p providing evidence against H_0.

TABLE I: Shapiro–Wilk Normality Test Results (Training Data without Outliers)

Feature	W Statistic	p-value
Age	0.9162	**0.0000**
Height_cm	0.9964	0.1673
Weight_kg	0.9971	0.3412
Training_Hours_Per_Week	0.9891	**0.0002**
Matches_Played_Past_Season	0.9446	**0.0000**
Previous_Injury_Count	0.8876	**0.0000**
Knee_Strength_Score	0.9977	0.5403
Hamstring_Flexibility	0.9990	0.9822
Reaction_Time_ms	0.9967	0.2410
Balance_Test_Score	0.9951	**0.0450**
Sprint_Speed_10m_s	0.9974	0.4441
Agility_Score	0.9964	0.1826
Sleep_Hours_Per_Night	0.9981	0.7178
Stress_Level_Score	0.9978	0.5882
Nutrition_Quality_Score	0.9967	0.2443
BMI	0.9955	0.0689

Table I summarizes the Shapiro–Wilk results after outlier removal, highlighting features that deviate from normality ($p < 0.05$) and those that approximately satisfy the Gaussian assumption. It is also notable that the Nutrition Quality Score and BMI features rejected the null hypothesis of normality before outlier removal but failed to reject it after outliers were removed, indicating that their earlier deviation from normality was primarily driven by extreme values.

3) Conditional Distribution Analysis: To study the relationship between each feature and injury occurrence, conditional distributions were plotted for each quantitative feature given the injury class. Kernel Density Estimation (KDE) plots, as shown in Figs. 7 and 8, were used to visualize the feature distributions for injured players (Class = 1) and non-injured players (Class = 0). These plots were generated using standardized feature values to ensure consistent scaling across features.

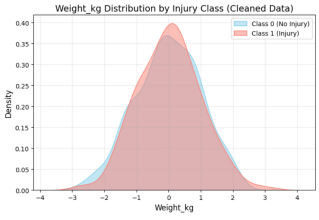

Fig. 7: Conditional KDE for a weakly discriminative feature.

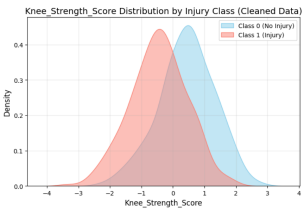

Fig. 8: Conditional KDE for a strongly discriminative feature.

The conditional distributions reveal that some features have little impact on injury prediction. For example, the distributions of *Weight* overlap almost completely between Class 0 (no injury) and Class 1 (injury), indicating minimal effect. In contrast, *Knee Strength Score* shows a slight shift, with higher scores more associated with Class 0, suggesting stronger knee strength reduces injury likelihood.

D. Gaussian Naive Bayes Implementation from Scratch

A Gaussian Naive Bayes (GNB) classifier was implemented entirely from scratch without using any machine learning libraries such as `scikit-learn`. Only basic numerical libraries, `NumPy` and `Pandas`, were used for mathematical operations and data handling. This implementation ensures a clear understanding of the underlying statistical principles of the Naive Bayes algorithm.

1) Overview of the Naive Bayes Classifier: Naive Bayes is a probabilistic classification algorithm based on Bayes' Theorem, with the assumption that all features are conditionally independent given the class label. Since the dataset contains continuous features, the Gaussian Naive Bayes variant was adopted, where each feature is assumed to follow a normal (Gaussian) distribution within each class. The classifier predicts the class c that maximizes the posterior probability:

$$P(c \mid \mathbf{x}) \propto P(c) \prod_{i=1}^{n} P(x_i \mid c)$$

To avoid numerical underflow, all probability computations are performed in the logarithmic domain.

2) Model Structure: The classifier was implemented as a Python class named `GaussianNaiveBayes`. The implementation consists of three main components:

- `fit()` for training the model
- `gaussian_log_pdf()` for computing Gaussian log-probabilities
- `predict()` for predicting class labels

3) Training Phase: During training, the model estimates the statistical parameters required for classification.

a) Class Identification: The unique class labels are extracted from the target vector y:

$$\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$$

b) Feature Statistics Estimation: For each class $c \in \mathcal{C}$, the mean and variance of each feature are computed as previously explained. These statistics are stored for each class and used during prediction.

c) Prior Probability Calculation: The prior probability of each class is calculated as:

$$P(c) = \frac{N_c}{N}$$

where N_c is the number of samples belonging to class c, and N is the total number of samples.

d) Zero-Variance Handling: To prevent division by zero in the Gaussian probability density function, any feature variance equal to zero is replaced by a small constant value:

$$\sigma_c^2 \leftarrow 10^{-9}$$

4) Gaussian Log Probability Density Function: The likelihood of a feature value given a class is computed using the Gaussian probability density function in logarithmic form:

$$\log P(x \mid c) = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{(x-\mu)^2}{2\sigma^2}$$

This formulation improves numerical stability and allows efficient summation across features.

5) Prediction Phase: For each test sample $\mathbf{x}$, the posterior log-probability for each class is computed as:

$$\log P(c \mid \mathbf{x}) = \log P(c) + \sum_{i=1}^{n} \log P(x_i \mid c)$$

The predicted class $\hat{c}$ is selected according to:

$$\hat{c} = \arg\max_{c \in \mathcal{C}} \log P(c \mid \mathbf{x})$$

6) Key Characteristics of the Implementation:

- No built-in machine learning classifiers were used.
- All statistical computations were implemented manually.
- Logarithmic probabilities were used for numerical stability.
- The model strictly follows the assumptions of Gaussian Naive Bayes.

E. Gaussian Naive Bayes Using `scikit-learn`

In addition to the custom implementation, a Gaussian Naive Bayes classifier was implemented using the `GaussianNB` class from the `scikit-learn` library. This implementation serves as a benchmark to validate the correctness and performance of the from-scratch model under the same experimental conditions.

1) Model Configuration and Training: The `GaussianNB` classifier was initialized with default hyperparameters. The model was trained using the same training dataset used in the previous implementation $(X_{\text{train}}, y_{\text{train}})$ by calling the `fit()` method. It was trained and tested on both the raw and cleaned datasets. The algorithm internally estimates the class priors, as well as the mean and variance of each feature for every class, following the Gaussian Naive Bayes formulation.

2) Prediction: After training, class labels for the test dataset (X_{test}) were predicted using the `predict()` method.

3) Model Evaluation Metrics: The performance of the scikit-learn Gaussian Naive Bayes model was evaluated using multiple standard classification metrics:

- **Accuracy:** Measures the proportion of correctly classified instances over the total number of samples. It is defined as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP, TN, FP, and FN denote true positives, true negatives, false positives, and false negatives, respectively.

- **Classification Report:** Provides class-wise evaluation metrics including precision, recall, and F1-score, computed as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Confusion Matrix:** A tabular representation of classification outcomes that summarizes the number of correct and incorrect predictions. For binary classification, it is expressed as:

$$\begin{pmatrix} TN & FP \\ FN & TP \end{pmatrix}$$

These metrics allow a comprehensive comparison between the library-based implementation and the custom Gaussian Naive Bayes classifier later shown in the Results section.

4) Purpose of Using `scikit-learn`: The scikit-learn implementation was used to:

- Verify the correctness of the from-scratch implementation.
- Provide a standardized baseline for performance comparison.
- Ensure reproducibility and alignment with established machine learning practices.

III. Results and Analysis

A. Implementation Comparison

The performance of the Gaussian Naive Bayes classifier implemented from scratch and the scikit-learn implementation was compared under identical experimental settings. When trained and evaluated on the complete dataset consisting of 800 samples, both models achieved the same classification accuracy of **0.975**.

After applying outlier removal, the accuracy of both models decreased to **0.9487**. This reduction in performance can be attributed to the class distribution of the removed samples. Specifically, the majority of the removed instances belonged to the positive injury class (target value = 1). As a result, the estimated prior probability of the injury class was reduced during training, which directly affected the posterior probability estimates produced by the Gaussian Naive Bayes classifier.

Since Naive Bayes relies heavily on class prior probabilities, altering the class balance by disproportionately removing positive-class samples leads to a bias toward the negative class. Consequently, this shift in class distribution explains the observed decrease in overall classification accuracy for both implementations.

TABLE II: Key Performance Metrics of Gaussian Naive Bayes on the Full Dataset (800 Samples)

Metric	From-Scratch NB	SKLearn NB
Accuracy	0.975	0.975
Precision (Class 1)	0.987	0.987
Recall (Class 1)	0.9625	0.9625
F1 Score (Class 1)	0.9747	0.9747

This identical performance clearly shown in Table 2 confirms the correctness of the custom implementation and demonstrates its consistency with the established library-based approach.

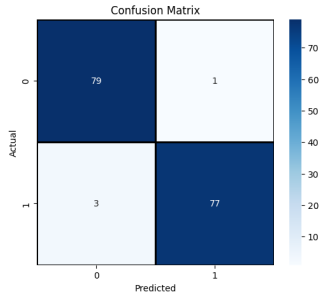

Fig. 9: Confusion matrix for the SKlearn model.

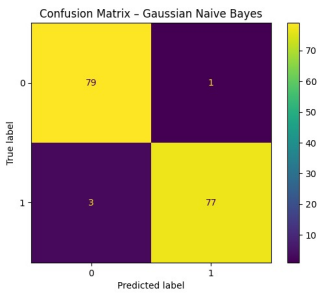

Fig. 10: Confusion matrix for the self-implemented model.

The confusion matrices shown for both models in Fig.9-10 indicate very few misclassifications, with only three false negatives and one false positive in each model.

TABLE III: Comparison of Gaussian Naive Bayes Performance After Outlier Removal (24 Samples)

Metric	From-Scratch NB	SKLearn NB
Accuracy	0.9487	0.9487
Precision (Class 1)	0.972	0.972
Recall (Class 1)	0.921	0.921
F1 Score (Class 1)	0.946	0.946
Confusion Matrix [TP / FP / FN / TN]	70/2/6/78	70/2/6/78

When analyzing the results for the cleaned dataset shown in Table 3, we notice:

- Accuracy remains the same for both models.
- The drop in recall for the positive class compared to the full dataset (from $0.9625 \rightarrow 0.921$) is consistent with the **removal of many positive-class outliers**, which reduces the prior probability of class 1.

IV. Conclusion

This study examined injury prediction in university football players using a combination of statistical analysis and Gaussian Naive Bayes modeling. Outlier detection using the IQR method identified 24 extreme observations, primarily in Matches Played and Sprint Speed, and their removal resulted in tighter feature distributions and improved Gaussian-like behavior for variables such as Nutrition Quality Score and BMI. Shapiro–Wilk tests further confirmed that outlier removal improved adherence to normality assumptions, especially for features that initially deviated from Gaussian behavior. Conditional distribution analysis revealed that some features, like Weight, had minimal predictive effect, while others, such as higher Knee Strength Scores, were associated with a lower likelihood of injury. Both the from-scratch and scikit-learn Gaussian Naive Bayes classifiers achieved high accuracy (0.975) on the full dataset, with a slight decrease to 0.9487 after outlier removal due to the disproportionate removal of positive-class samples. Precision, recall, and F1-score metrics remained robust for both models, and the close agreement between the custom and library implementations validates the approach and demonstrates reliable performance, with very few misclassifications. These findings suggest that features with strong class separation, like Knee Strength Score, can guide monitoring and preventive strategies, while weakly discriminative features may be deprioritized to streamline data collection and modeling. Overall, the results provide clear insights into factors affecting injury risk and demonstrate that Gaussian Naive Bayes can effectively support injury prediction in university football players.

V. Member Contributions

- **Mona – Data Handling & Preprocessing:** Managed the dataset by acquiring, cleaning, and preparing it for modeling. Performed feature engineering, computed descriptive statistics, handled missing values and outliers, and executed an $80\%/20\%$ train-test split.
- **Khadija – Statistical Analysis & Visualization:** Validated statistical assumptions for Naive Bayes by analyzing feature distributions, conducting normality tests, defining hypotheses, interpreting p-values, and plotting conditional distributions to assess feature relevance.
- **Rowida – Naive Bayes Classifier (From Scratch):** Developed the Gaussian Naive Bayes algorithm from first principles, including class prior estimation, Gaussian PDF implementation, posterior probability computation, and evaluation of model accuracy.
- **Malak – Library Model & Model Comparison:** Implemented the `sklearn GaussianNB` model, computed performance metrics, compared results with the manual implementation, and finalized all figures, tables, and outputs for the report and presentation.

References

[1] Yuanchunhong, *University Football Injury Prediction Dataset*, Kaggle, 2023. Available at: https://www.kaggle.com/datasets/yuanchunhong/university-football-injury-prediction-dataset/data

[2] Tukey, J. W. (1977). *Exploratory Data Analysis*. Addison-Wesley.

[3] Bobbitt, Z. (2021, November 15). *The Complete Guide: When to Remove Outliers in Data*. Statology. https://www.statology.org/remove-outliers/

[4] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3–4, pp. 591–611, 1965.