Predicting Table Tennis Experience from 9-Axis Racket Sensor Data

Abstract:

This report investigates the effectiveness of classifying player experience in table tennis using only racket-mounted 9-axis motion sensor data. Using a dataset of 90,000 swings and applying three classification models, Logistic Regression, Random Forest and XGBoost, this claim was assessed. XGBoost proved to have the highest performance (89% accuracy), suggesting that non-linear models are best to capture the relationships between swing data and playYears. This study highlights and evaluates the benefits and limitations of each model.

Introduction:

Table tennis is a popular sport played around the world with millions of people having played it. Naturally, like other sports, there are a range of diverse skills you can develop and techniques you can employ. In the report, the following question will be addressed, “Can motion sensor swing data alone classify player experience?”

The dataset that will be handled is named “TTSWING” and contains swing information obtained from 9-axis sensors integrated within the racket. The original dataset examines 90 table tennis players of varying experience, containing over 90,000 swing information (Chou et al., 2023).

Data Preprocessing:

Before modelling, the data must first be cleaned and pre-processed such that the later models can achieve a reliable result. The first step employed was to remove any irrelevant features. This includes any demographic columns like age, gender, handedness, and any metadata columns that do not help in answering the research question. Missing data was not a problem. However, there were some rows that were invalid (i.e. containing all 0’s), and this was removed. Furthermore, all categorical variables such as the target variable PlayYears, has been binary encoded. For example, the original PlayYears was classified with either (High, Medium and Low), and this was changed to (2, 1, 0). Outliers were also considered, where any rows that have 8 or more outlier values were removed based on IQR. The IQR method was chosen over z-score due to its effectiveness in non-gaussian distributions. The last step was to check for consistency between all datatypes, and this was not a problem. After all cleaning methods were performed, 39 columns were left with 87101 rows.

Once cleaning was finished, some basic EDA was performed. An area of concern was to verify the class balance of the target variable ‘playYears’.

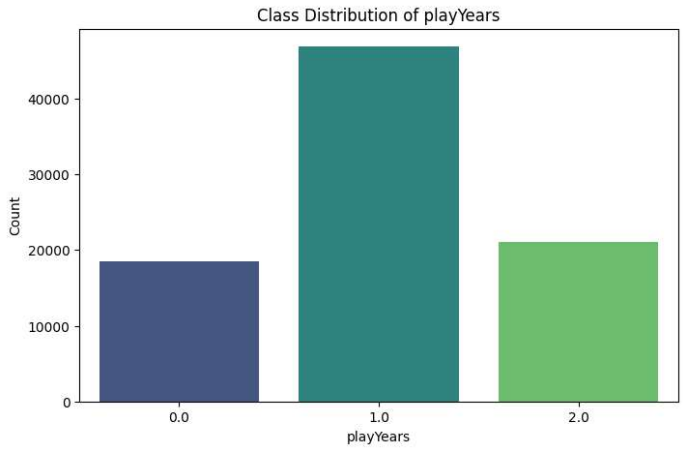


Figure 1: class distribution of ‘playYears’

As highlighted above, there is a moderate class imbalance where players of medium experience dominate the dataset comparatively towards low and high experience players. Although techniques like SMOTE are commonly used to address this problem, this was not opted for use for a few key reasons (Satpathy, 2021):

1. Data Authenticity
   SMOTE generates synthetic rows for the minority class by interpolating between the existing minority data. In this report, where player experience may involve complex and non-linear relationships, introducing artificial samples can potentially distort the data distribution or introduce unrealistic patterns.
2. Model Class imbalance handling
   Instead of altering the data, the report opted to use simple sampling methods instead by setting class_weight = 'balanced' in all models. This automatically adjusts weights inversely proportional to class frequencies, helping the model pay more attention to the minority class.

Data skewness was a further area of interest due to its impact on regression models performance and interpretability. Skewed features can violate the assumptions of regression models, such as the assumption of normally distributed residuals, and may lead to incorrect coefficient estimates or worse model performance.

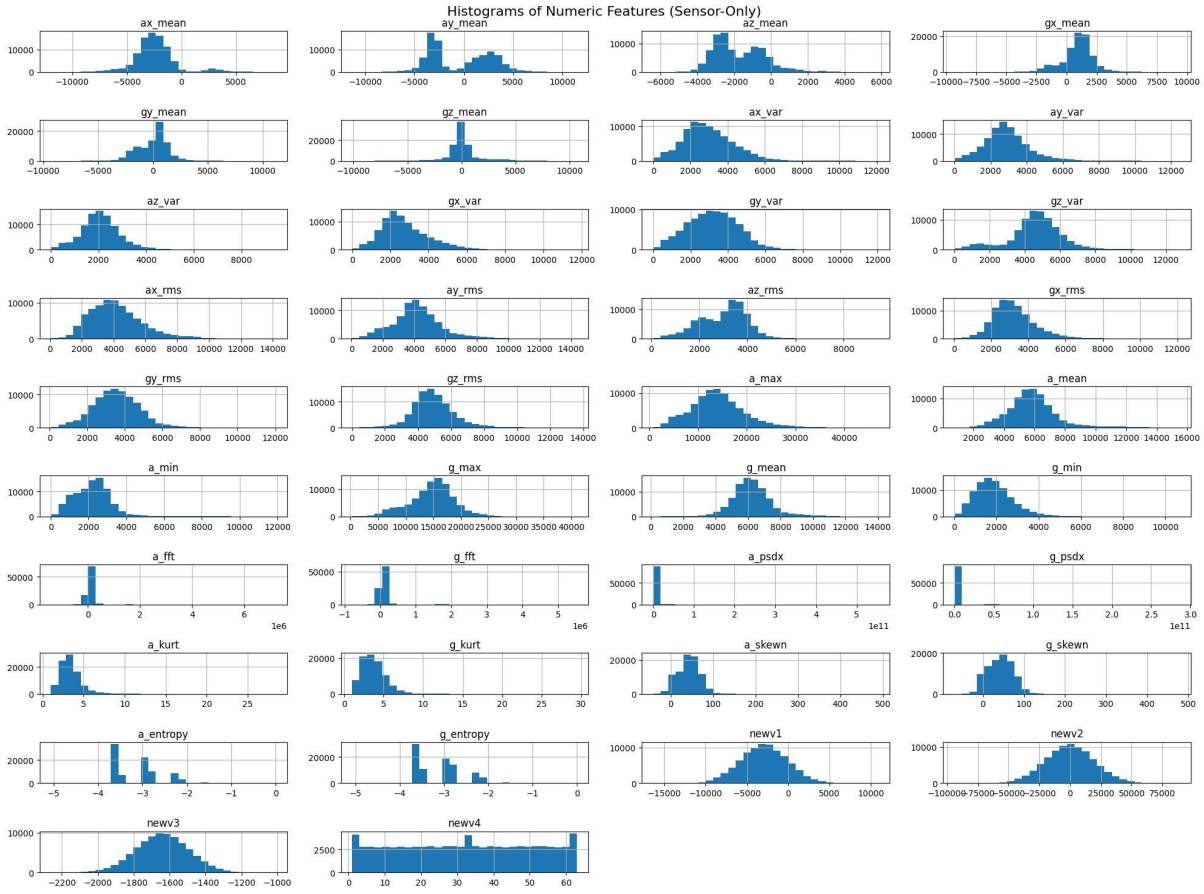


Figure 2: Data distribution of all input features

During EDA, all features' data distribution was assessed using histograms for significant skewness. Fortunately, all features display a level of normal distribution, and thus, no transformations (e.g. log, Box-Cox) were required to correct for skewness.

Multicollinearity was also investigated due to its known impact to negatively influence regression models. To assess, a correlation heatmap matrix was used between all input features.

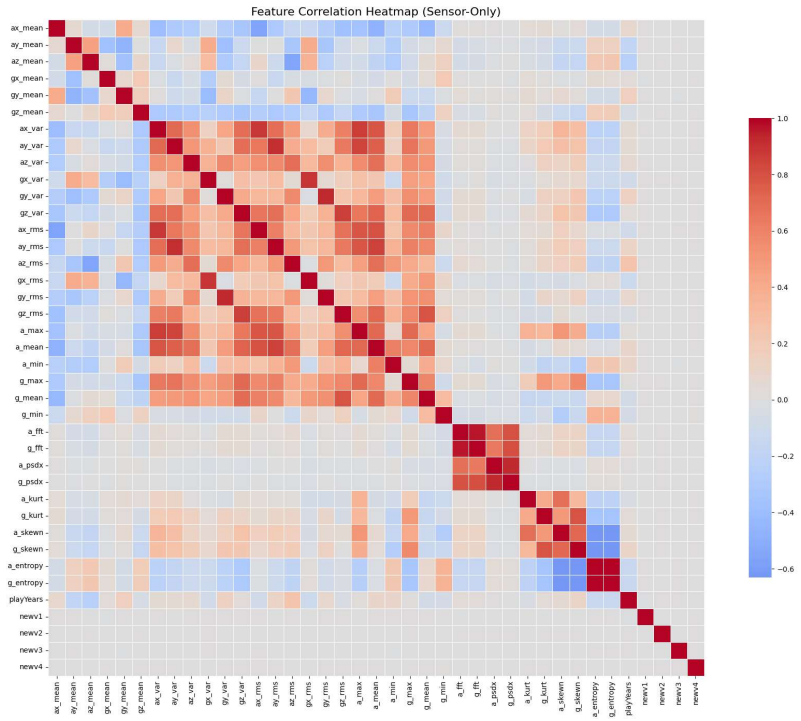


Figure 3: Correlation heatmap matrix between all variables

Note: as a rule of thumb, a correlation value of ±0.7 or more is considered high

As expected, most features of the sensor data are highly correlated with each other. This is natural to see in table tennis, where motion-based sensors on the racket tend to move together as a part of a swing. While this high degree of correlation does not immediately pose a problem, it can affect feature importance interpretability of certain models. For further research, PCA could be applied. However, this also has its limitations in feature importance.

Metrics:

To assess models and ensure a level of interpretability and comparatively, the following metrics were used (google, 2024):

Accuracy: The proportion of correct predictions out of total predictions. Very useful in interpretability, though it may prove a bit misleading when used with an imbalanced dataset like TTSwing.

Precision: the ratio of true positives across all predicted positives

Recall: the ratio of true positives across all actual positives

F1-score: the primary metric used to evaluate models. It represents the harmonic mean between precision and recall. This is extremely informative for imbalanced datasets like TTSwing

ROC Curve and AUC: plots the trade-off between true positive rate and false positive rate. AUC highlights the overall model discrimination ability in classification tasks.

Confusion Matrix: Shows the distribution of correct and incorrect predictions for all classes. Assists with identifying where the misclassification lies.

For all metrics, macro averaging was used to ensure that each class contributed equally, addressing the moderate class imbalance present in the dataset.

Model Training:

To predict a player's 'playYears' from sensor only features, this report explored 3 different machine learning algorithms. More specifically, Logistic Regression, Random Forest and XGBoost. Each model offers a different learning algorithm and perspective in assessing which model performs the best on this dataset.

1. Logistic Regression

   Serves as a baseline model. It is fast and easily interpretable. It assumes linear decision boundaries, making it perfect for datasets where classes are well-separated in feature space. Additionally, it is fast and has low computational cost (geekforgeeks, 2023).

2. Random Forest

   A powerful ensemble method that uses multiple decision trees and combines their outputs to improve generalization. This would prove useful in handling non-linear feature relationships (i.e. feature sensors). It also handles multiclass classification innately and mitigates the risk of overfitting well. More importantly, it is robust against outliers and noise (IBM, 2021).

3. XGBoost

   A very powerful gradient boosting algorithm well regarded for its predictive power, especially on tabular structured data. It builds models sequentially, with each iteration aiming to reduce the error of its ancestors. A benefit of this model comes from its extensive parameter choices, and the ability to fine tune such variables. Like random forest, it can capture non-linear relationship very well and is robust against outliers (Nvidia, n.d.).

For the first step, standardizing the data using 'StandardScaler()' was applied for all models. This ensures all data has mean 0 and standard deviation 1. Some of the benefits include improved model convergence, fair feature contribution and improved regularization.

Due to the risk of overfitting seen in some of these models and to assist with hyperparameter tuning, 5-way cross validation was applied to all 3. 5 was chosen as the optimal k-values due to its performance seen in previous research and it offers low bias and variance without being too computationally expensive.

Because of the moderate class imbalance seen in playYears, 'classweight = balanced' was applied towards the models to adjust the weights inversely proportional to class frequencies, ensuring that the minority classes received more attention during training and helping to reduce bias toward the majority class.

To ensure there is no data leakage, XGBoost, Random Forest and logistic regression received a training test split of 80-20. Additionally, all models were packaged in a pipeline to prevent data leakage and ensure a streamlined workflow.

For hyperparameter tuning, human error was removed by using GridSearchCV to systematically explore all hyperparameter combinations for random forest and XGBoost. This automated tuning helps find the optimal settings for each model and helps against overfitting by evaluating performance on multiple data splits. For the scoring metric, F1-macro was used to ensure a balanced comparison between all models, which is especially important in moderately imbalanced datasets like TTSwing.

Assumptions:

Regarding the original dataset that contains 90 players and 90000 recorded swings, it was assumed that this dataset was sufficient in representing the player base globally. It was also assumed that the players in this dataset were diverse enough in skill, age, and physical attributes to generalize the model's performance beyond the training sets.

Another major assumption made was that the GridSearchCV technique will be able to find the optimal hyperparameter combination for Random Forest and XGBoost. This ultimately assumes that the chosen parameter grids are sufficiently comprehensive and well-structured to explore all combinations.

It is assumed that the 90,000 recorded swings are independent observations, even though the swings may comes from the same player, session or conditions. This is important as many ML models rely on the idea that each observation provides unique information. Using this assumption of independence, makes it easier to train and evaluate models. However, it risks overestimating model performance if the model overfits player-specific patterns.

Logistic Regression:

The Sci-kit learn documentation outlines 16 hyperparameters for Logistic Regression. However, to serve as a base model, a simple multinomial logistic regression model was implemented with no regularization (scikit-learn, 2014). Below showcases the hyperparameter choices:

Table 1: Logistic regression model hyperparameter values

Hyperparameter	**Value**
penalty	None
solver	'lbfgs'
multi_class	'multinomial'
class_weight	'balanced'
max_iter	1000

Note: solver was 'lbfgs' due to its speed and reliability, especially on multinomial models. Max_iter was set at 1000 as it balanced speed and is more than sufficient for the model to converge and ensure early stopping.

Using these hyperparameters, the following results are shown:

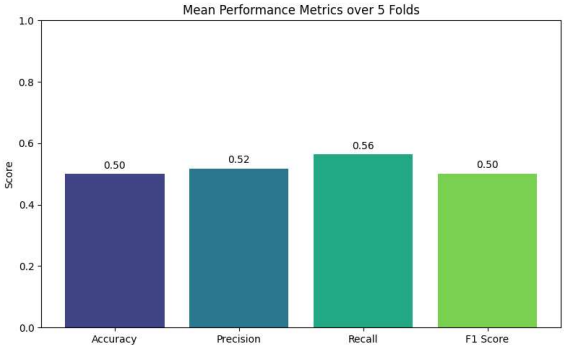


Figure 4: Logistic metric results

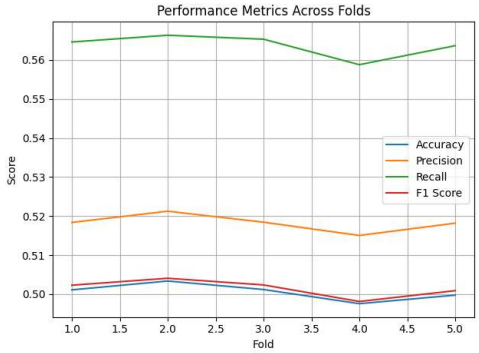


Figure 5: Logistic performance across folds

As observed above, the model achieved a mean accuracy of 0.5, which is slightly better than random guessing (~0.33). The mean precision (0.52), recall (0.56) and f1 score (0.50) further showcase the models poor predictive performance. These results suggest that the model is likely underfitting the data, where the model is not learning all the relationships between the features and the target variable. This is further reinforced by the consistency of these 4 metrics across all 5 folds, which demonstrates the model's

reliability and consistent poor performance. To dive further, the confusion matrix and ROC curve is modelled:

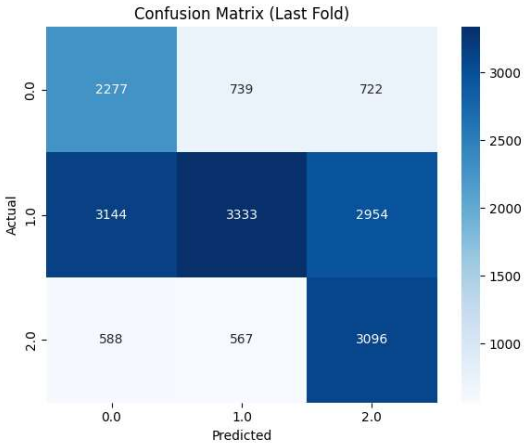


Figure 6: Logistic confusion matrix

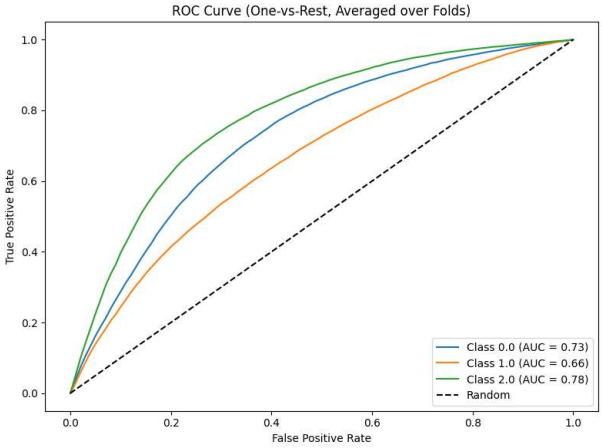


Figure 7: Logistic one vs rest ROC curve

In the confusion matrix, it is evident where the problem lies. For class 0, there is a ~61% accuracy rate, implying that class 0 can be distinguished moderately decently. In class 1, the number of correct predictions is ~35.3% with predictions spread out evenly between the other classes (i.e. random guessing). This effectively highlights a severe class overlap and poor feature separability. Comparatively, class 2 performs the best with a ~72.8% accuracy. The following ROC curves support this argument where class 2 (AUC = 0.73) performs the best compared to class 1 (AUC = 0.66) and class 0 (AUC = 73).

The effects of applying regularisation will now be investigated. To do so, GridSearchCV will be used to conduct the optimal hyperparameter combination between L1, L2 and elastic regularisation (L1 &L2), where ideally it would find the best penalty technique and penalty rate. This process involves defining a range of regularisation strengths using a list of lambda values which are then inversed to get penalty rate 'c'. In total, 25 different c-values were explored for each regularization strength, and the solver was set to 'saga' as it supports all regularisation methods.

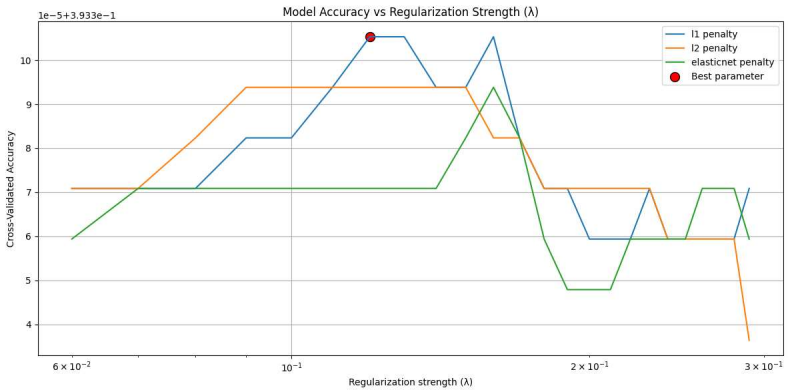


Figure 8: Effects of regularisation on Logistic regression

This plot demonstrates how model accuracy varies with different regularization strengths for 3 different kinds of strategies, L1, L2 and ElasticNet. As lambda increases, it is illustrated that cross validated accuracy generally declines. The best performance was seen in L1 penalty with a c value of ~8.33. This suggests that the L1 penalty, which promotes sparsity by zeroing out less important features, leads to better generalization in this case.

Random Forest:

Out of the 20 potential tuneable hyperparameters, 4 parameters were adjusted. Again, this selection was made based on a grid search and the impact each parameter has on the results based on previous research (scikit-learn, 2025).

Table 2: Random forest hyperparameter choice

Hyperparameter	**Values Tested**	**Optimal value**
n_estimators	[100, 150, 200]	200
max_depth	[10, 20, 30]	30
min_samples_split	[2, 5, 10]	10
min_samples_leaf	[1, 2, 4]	4

By running grid search, a total of 81 different parameter combinations were tested. In total 405 model fits were made (i.e. 5-fold cross validation).

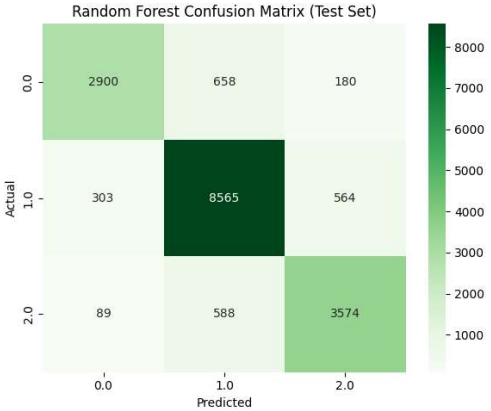


Figure 9: RF confusion matrix

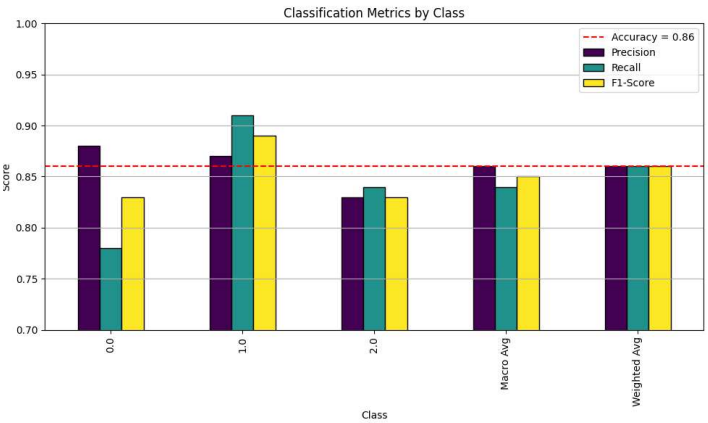


Figure 10: Performance metrics for RF

A noticeable improvement from the logistic regression is displayed from the confusion matrix and metrics. Accuracy reached a score of 86% with metrics such as macro average scores for precision, recall and f1-scores ranging from 0.84 to 0.86.

In the confusion matrix, it is apparent that most test samples have been properly identified. Furthermore, most errors are due to confusion between neighbouring classes 0, 1 and 1, 2, which is to be expected.

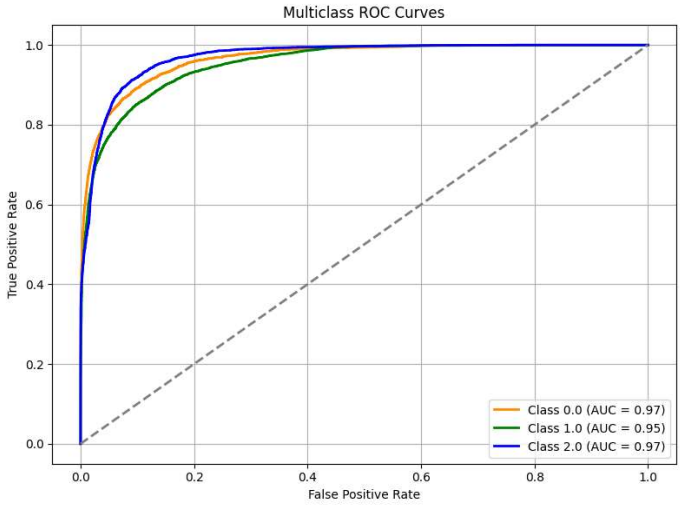


Figure 11: One vs rest ROC curve for RF

From the ROC curves and AUC scores, all AUC are above 0.95 which indicate very good model discrimination for all classes. The curve hugs the top-left corner, suggesting low false positive and high true positive rates. Class 1 has a slightly lower AUC, thereby, highlighting that class 1 feature discrimination overlaps with other features.

An interesting aspect of tree-based models are that they are non-deterministic. More specifically, due to the bootstrapping/random splits, model performance can vary depending on the seed. To address this, the report will:

1. Repeat the training and evaluation of the optimal model 10 times using different seeds
2. Record metrics such as accuracy and f1-score

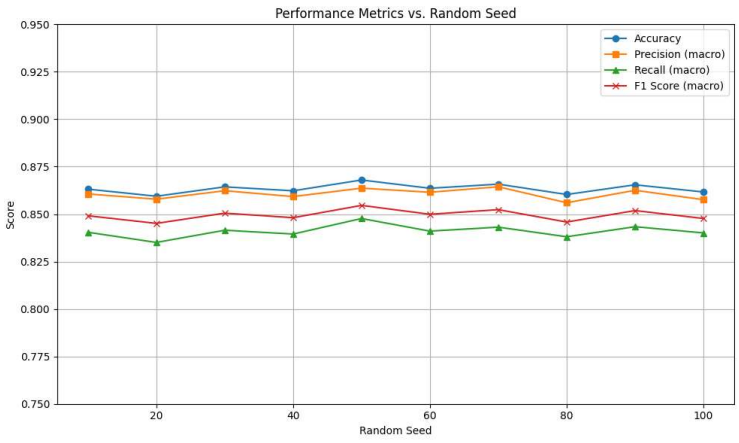


Figure 12: Random seed on random seed performance metrics

From the above, it is easy to see that there are little fluctuations between seed values for the key metrics. Hence, it can be concluded that the variance of Random Forest is quite low.

This report decided to further investigate the effects of max_depth and min_samples_leaf due to their effect on controlling complexity, performance and generalization ability of decision tree models. Extreme values of both low and high magnitudes were chosen (i.e. max_depth_range = [1, 5, 10, 20, 30, 50, None] and min_samples_leaf_range = [1, 2, 4, 10, 20]).

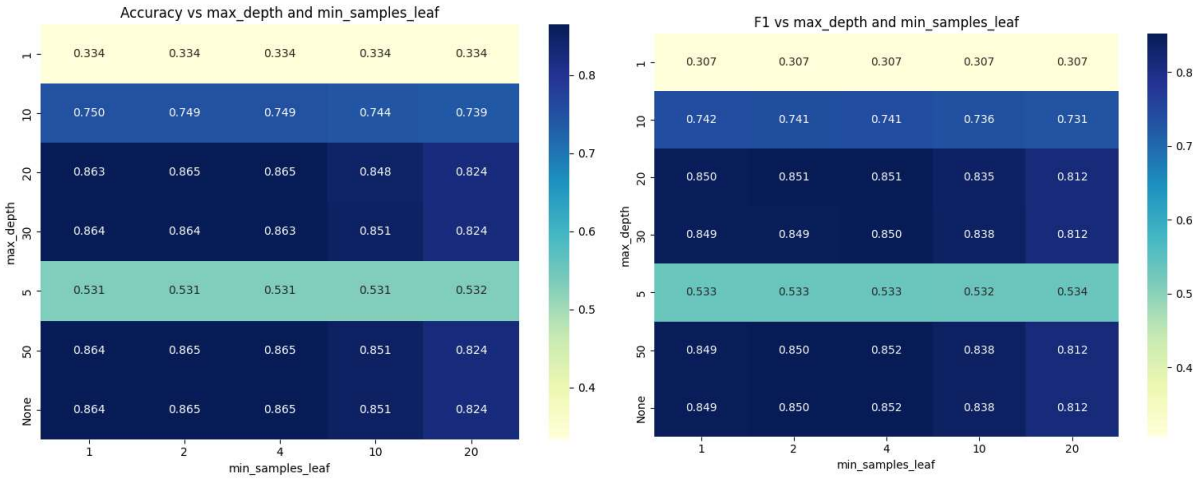


Figure 13: accuracy and F1-score vs max_depth and minsamples_leaf

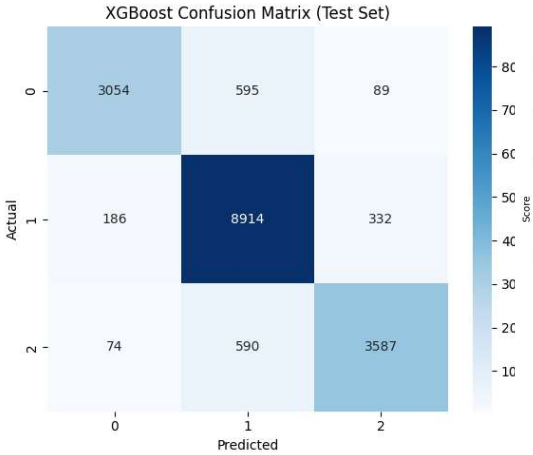
The most promising performance occurs at max_depth 20, 30, 50, None, with min_samples_leaf set to 2 or 4. In these cases, accuracy and F1 score achieves its highest value of ~ 0.865 and ~ 0.851. Increasing min_samples_leaf to 10 or 20 reduces accuracy and f1 score slightly. Finally, the parameter None does not outperform depths of 20 or 50, suggesting that there is little value in growing deep trees. Reducing tree depth can, therefore, reduce overfitting and run time while keeping metrics high.

XGBoost

XGBoost's documentations highlights 23 tuneable hyperparameters. But in practice, only a subset of parameters significantly impacts model performance. For this analysis, 5 of the most influential attributes will be chosen based on their proven influence on model complexity, performance and training speed. The rest of the parameters will be left to their default value. In total, 108 different hyperparameter combinations were tried, for a total of 540 different model fits (xgboost, 2022).

Table 3: XGBoost hyperparameter choice

Hyperparameter	Values Tested	Optimal Value
xgb_n_estimators	[150, 250, 350]	350
xgb_max_depth	[6, 10, 15]	15
xgb_learning_rate	[0.1, 0.3, 0.5]	0.3
xgb_subsample	[0.8, 1.0]	1.0
xgb_colsample_bytree	[0.8, 1.0]	0.8



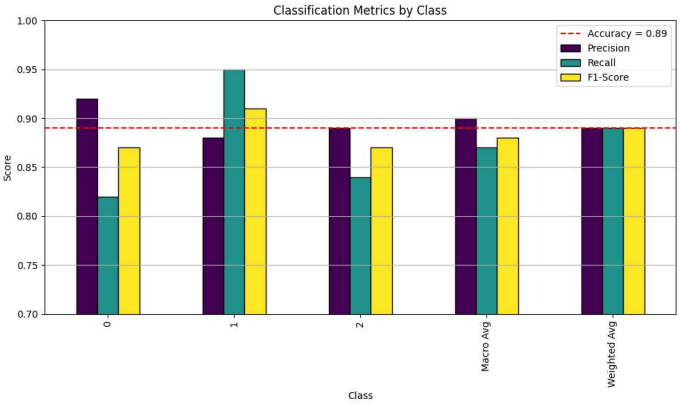


Figure 14: XGBoost confusion matrix Figure 15: XGBoost performance metrics

A slight improvement from random forest, XGBoost presents an accuracy score of 89%. Metrics such a precision, recall and F1-score are also consistently high with scores generally ranging from 0.87% – 0.95%. The confusion matrix reinforces this high performance, where 82.1%, 94.8%, 84.8% are captured for classes 0, 1, 2 respectively. Like the random forest, most inaccuracies source from the misclassification of neighbouring classes of 1.

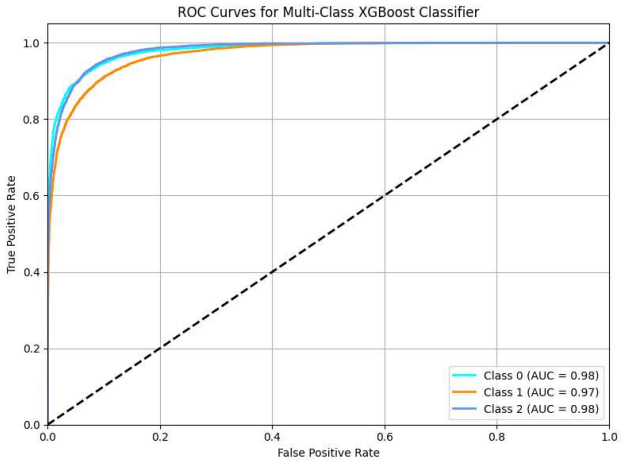


Figure 16: One vs rest ROC Curve for XGBoost

Looking further into the ROC curve, the model performs exceedingly well in feature discrimination with scores all above 0.97. Below, an experimentation of the seed values and the non-deterministic nature of XGBoost will be explored.

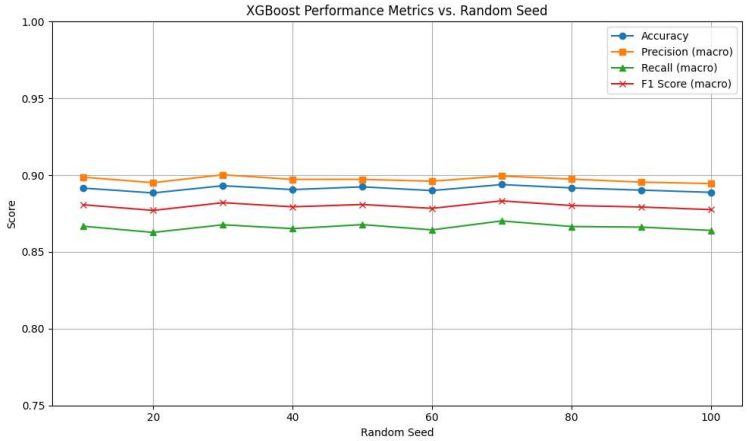


Figure 17: Random seed vs XGBoost performance

Here, there is even less variation compared to random forest. Fluctuations are very minor with an error range of 0.02 with each seed sample.

To further refine the model and prevent overfitting, additional experimentation was conducted on hyperparameters gamma and regularization variables. The gamma parameter effects the minimum loss reduction required to make a further partition on a leaf node of the tree. By increasing gamma, ideally the model becomes more conservative in adding splits (i.e. helps reduce overfitting on noisy data).

Regularization parameters L1 and L2 were also explored in their ability to apply penalties to XGBoost. Tuning these parameters helps control model complexity, improve model generalization and reduce overfitting.

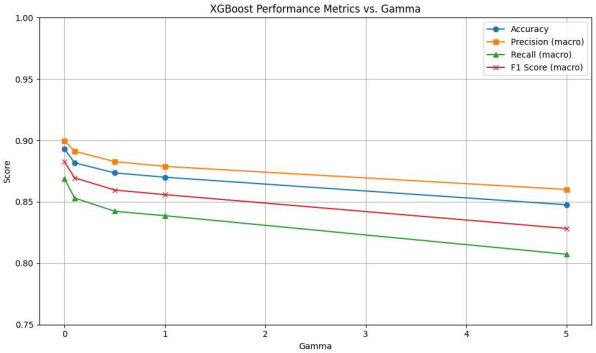


Figure 18: XGBoost metrics vs Gamma

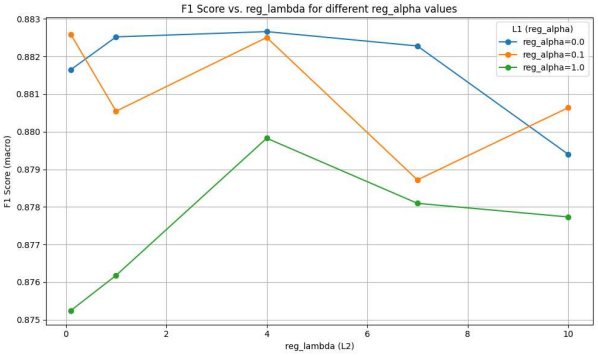


Figure 19: F1 vs regularisation

In the first graph, all metrics decline as gamma increases. The sharpest decline is seen from gamma = 0 to 1. Hence, it can be concluded that the best performance occurs when no gamma is added. For the regularization figure, it is evident that a small amount of L1 regularization (i.e. 0.1) with moderate L2 (i.e. 4) and no L1 regularization with

moderate L2 is optimal. Higher L1 values of 1.0 hurt performance, likely due to the over-penalization of weights which could lead to underfitting. Overall, this suggests that moderate regularization is beneficial, but over tuning regularization can generally lead to the degradation of model generalization.

Comparison and Discussion

To analyse the performance of all models, all metrics (macro) are displayed in a single table and graph.

Table 4: Performance metrics across all models

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.50	0.52	0.56	0.50
Random Forest	0.86	0.86	0.84	0.85
XGBoost	0.89	0.90	0.87	0.88

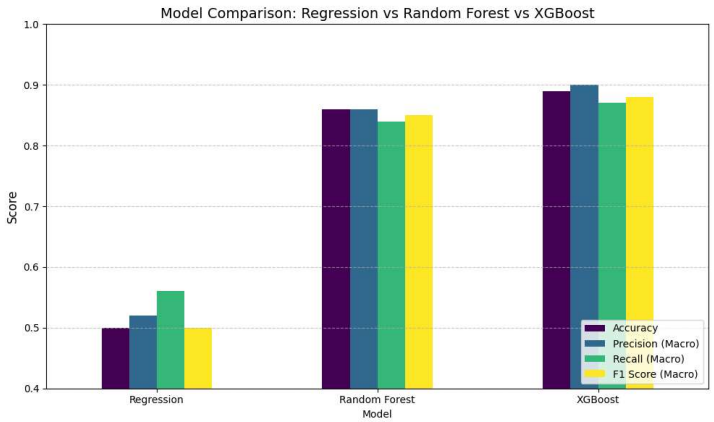


Figure 20: metric comparison across all models

From a single glance, it is clear XGBoost performs the best across all models. All metrics from XGBoost range from 0.87 to 0.90. Random forest features comparative performance with metrics of range 0.84 to 0.86. In contrast, logistic regression performed significantly worse compared to the other models, where all metrics fall below 0.6.

In terms of bias-variance trade-off, logistic regression suffers from high bias and low variance, causing it to underfit complex datasets as highlighted above. Random forest and XGBoost offer lower bias due to their inherent nature to model non-linear relationships, with additional mechanisms to keep variance low.

XGBoost's superior performance can likely be sourced from its boosting mechanisms. The inherent structure such that trees are built sequentially, where each tree aims to reduce the errors of its previous predecessors, likely leads to the greater performance compared to random forest. Additionally, the inbuilt regularization measures help reduce overfitting and reduce variance, which is a common issue seen in ensemble methods. Moreover, the wide range of tuneable hyperparameters and features like tree pruning and parallelized tree construction makes it highly efficient. This is most likely what leads to the high scores seen across all metrics.

Random Forest is an ensemble method like XGBoost but uses bagging instead of boosting. This method constructs multiple trees in parallel, employing bootstrapping to train. This reduces variance by averaging the trees; however, it does not actively try to correct the mistakes of other trees or reduce bias as effectively as XGBoost. Random forest may also include redundant or weak trees that XGBoost would otherwise prune or refine. Nonetheless, this technique performs well due to its ability to handle all non-linear relationships seen in the sensor data.

Perhaps the greatest model of interest is the logistic regression. Its poor performance can be attributed to its assumption of linear relationships between input features and the target variable, which is most likely too simplistic for sensor-data. Sensor-based datasets like TTSwing generally contains very non-linear relationships. For example, the effect of one motion-based data on 'playYears' is very dependent on the values of other features. Logistic regression struggles with multicollinear data unlike XGBoost and Random Forest, which could reduce predictive performance as the model becomes sensitive to small changes in data, contributing to both high bias and unreliable variance.

Another factor to consider is each models response to noise, a frequent and common issue seen in sensor data. Motion sensors can sometimes pick up variability due to inconsistent swings or problems related to the sensor itself. In random forest, the averaging of multiple trees reduces the impact of any one noise feature. XGBoost refines this process by focusing on each tree's residual errors, correcting this noise. However, logistic regression has no mechanism to handle noise. Thus, making it highly sensitive to multicollinearity and outliers, leading to poor generalization and low metric results.

A common problem seen in all models is that classes 0 and 2 are being misclassified into class 1. This is to be expected due to the inherent nature of the target variable 'playYears'. Class 0 is defined a low number of years played, thus translating to obvious patterns such as slower swing. While class 2 represents players of more extensive careers. Class 1 sits between both extremes, where players may play for a longer amount of time compared to class 0, but have not displayed the appropriate skills of such. In contrast, some class 1 players may display a level of prodigious talent and

display what a class 2 player would. This overlap in feature patterns between adjacent classes likely contributes to most misclassifications, as players near the threshold of all classes may share similar actions.

Conclusion

This study poses the question on whether sensor data can be used to classify players 'playYears'. The dataset was sourced from TTSwing and was pre-processed to later fit the models. 3 models were explored with varying successes, logistic regression, random forest and xgboost. Experimentation was conducted on all models regarding hyperparameter tuning, regularization and variability. XGBoost was concluded to be the best model with an accuracy of 89% and F1-score of 0.88. Random forest and logistic regression follow behind with accuracy and f1 scores of 0.86, 0.85 and 0.50, 0.50 respectively. Further investigation can include a more extensive grid search for hyperparameter tuning, additional feature engineering techniques (e.g. SMOTE, VIF), or exploration of deep learning models.

References

Chou, C.-Y., Chen, Z.-H., Sheu, Y.-H., Chen, H.-H., & Wu, S. K. (2023). *TTSWING: a Dataset for Table Tennis Swing Analysis*. ArXiv.org. https://arxiv.org/abs/2306.17550

geeksforgeeks. (2024, May 9). *Understanding Logistic Regression*. GeeksforGeeks. https://www.geeksforgeeks.org/understanding-logistic-regression/

google. (2024). *Classification: Accuracy, recall, precision, and related metrics*. Google for Developers. https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall

IBM. (2021, October 20). *Random Forest*. Ibm.com; IBM. https://www.ibm.com/think/topics/random-forest

SATPATHY, S. (2020, October 6). *SMOTE - A Common Technique to Overcome Class Imbalance Problem*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/

scikit-learn. (2014). *sklearn.linear_model.LogisticRegression — scikit-learn 0.21.2 documentation*. Scikit-Learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Scikit-Learn. (2025). *sklearn.ensemble.RandomForestClassifier — scikit-learn 0.20.3 Documentation*. Scikit-Learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

What is XGBoost? (n.d.). NVIDIA Data Science Glossary. https://www.nvidia.com/en-au/glossary/xgboost/

XGBoost Parameters — xgboost 1.5.2 documentation. (2022). Xgboost.readthedocs.io. https://xgboost.readthedocs.io/en/stable/parameter.html

Appendix

Code can be found in GitHub: https://github.com/GavinSaiun/Motion-Based-Table-Tennis-Skill-Prediction