# Application Project Milestone 3

Hongyi Zhang[1], Ziang Deng[1], and Nancy Wang[1]

Team Name: Ziang/Nancy/Hongyi

[1] *McKelvey School of Engineering, Washington University in St. Louis, USA*
*hongyi.zhang@wustl.edu, d.ziang@wustl.edu, nancy.wang@wustl.edu*

# 1 Introduction

In this milestone, we participated in a Kaggle competition focused on a regression task evaluated by the Root Mean Squared Error (RMSE) metric. The objective was to build and train a machine learning model using a provided dataset, predict target values on an unseen test set, and compete against other teams and baselines based on predictive performance.

Our approach combined data exploration, systematic feature engineering, careful data preprocessing, and a well-designed artificial neural network model. Starting with extensive data cleaning, we ensured the integrity of the training and test datasets by checking for missing values, duplicates, and constant features. We also performed statistical analysis using Pearson correlation and p-values to identify and drop irrelevant features, reducing model complexity and potential overfitting.

For feature scaling, we adopted a customized scaling strategy tailored to different feature types. Specifically, we applied StandardScaler, MinMaxScaler, and RobustScaler to different groups of features, utilizing a ColumnTransformer to streamline the preprocessing pipeline. This preprocessing was critical to ensuring that the model could converge efficiently during training.

Our machine learning model was an artificial neural network (ANN) built with PyTorch, designed specifically for regression. The network architecture incorporated techniques such as L2 regularization, Batch Normalization and Dropout to improve generalization and prevent overfitting. To achieve optimal performance, we conducted a random search over key hyperparameters including learning rate, batch size, and weight decay followed by rigorous 5-fold cross-validation. The best hyperparameters were then used to retrain the model and generate final predictions.

Finally, ensemble techniques were used by averaging predictions across folds, leading to a more stable and robust final submission. All work, including preprocessing, modeling, evaluation, and prediction generation, was documented through cleanly organized scripts and a reproducible Kaggle notebook, adhering strictly to competition submission requirements.

This report details each step of the process, from data understanding and cleaning, through modeling choices and hyperparameter optimization, to the final results. We conclude with reflections on model performance, limitations, and potential improvements for future work.

# 2 Data Exploration and Milestone Recap

Throughout Milestone 1 and 2, we conducted a comprehensive exploration of the dataset and systematically evaluated a variety of preprocessing and feature engineering strategies in preparation for final model development.

In Milestone 1, we began with extensive exploratory data analysis (EDA), revealing that many featuresparticularly those related to timestampswere highly redundant and strongly correlated with each other. Kernel density estimates and correlation matrices helped identify both informative and irrelevant features. We also evaluated three baseline regression models: Ridge Regression, Random Forest, and a shallow Artificial Neural Network (ANN). The ANN achieved the best predictive accuracy, although at the cost of longer training time. We additionally performed K-means clustering (with $k = 2$) and found that while the distance-to-centroid features provided minor improvements for linear models, they failed to improve neural network performance.

In Milestone 2, we focused on feature selection and dimensionality reduction. Hand-selected features were chosen using correlation coefficients and p-value filtering, while dimensionality reduction was performed via PCA. We compared five feature configurations: raw features, hand-selected features, PCA with two components, PCA with optimal components (retaining $\geq 99.9999\%$ variance), and cluster-derived features. A consistent ANN architecture was used to evaluate each setting via 10-fold cross-validation and test set performance. Hand-selected and PCA-optimal features yielded the best performance, with mean squared errors (MSE) near 0.024 on the test set. Cluster-based and overly compressed PCA features performed significantly worse. These insights guided our decision to retain and refine the ANN model in Milestone 3 using the most informative features and a more robust training pipeline.

# 3 Data Preprocessing and Feature Selection

## 3.1 Dataset Description

In this project, we work with three main datasets: **training set**, **test set**, and **submission file**. Their structures and purposes are described as follows.

**Training Set** The **training set** is used for model development and contains both input features and the target variable.

- The dataset has **8250 samples** and **41 columns**.
- The first 40 columns correspond to various **input features**, such as `acc_rate`, `track`, `m`, `n`, `current_pitch`, ..., `omega`, `set`.
- The last column, `target`, represents the **variable** that we aim to predict.

**Test Set** The **test set** is used to evaluate the model's generalization ability. It contains only feature information without the target labels.

- The dataset includes **5500 samples** and **41 columns**.
- It consists of an `Id` column (a unique identifier for each instance, corresponding to the submission file) and the same 40 feature columns as in the training set.
- The target variable is not available in the test set.

**Submission File** The **submission file** is the required format for submitting predictions on Kaggle platform.

- It only contains two columns: `Id` and `target`.
- The `Id` matches the `Id` in the test set, and the `target` column must be filled with the model's predicted values.

## 3.2 Missing Values and Integrity Detection

Before proceeding with model training, we performed several sanity checks on the datasets, including missing value detection, duplicate row detection, and constant feature detection.

Prior to these checks, we preprocessed the original datasets by separating the features and target variable from the training set. And the `Id` column was removed from the test set. This step ensures that subsequent data processing and feature engineering operations can be uniformly applied.

**Dataset Size**

- Training set: 8250 samples, 40 features.
- Test set: 5500 samples, 40 features.

**Missing Values** To detect missing values, we computed the number of NaN entries in each feature. The results are summarized as follows:

- No missing values were found in the training set.
- No missing values were found in the test set.

Thus, no imputation or further handling of missing data is required.

**Duplicate Rows** We also checked for duplicate rows, which might introduce bias into the model:

- Training set: 0 duplicate rows.
- Test set: 0 duplicate rows.

Since no duplicate entries were found, no deduplication is necessary.

**Constant Features** Constant features (features with only one unique value across all samples) provide no useful information for prediction and should be removed. Our inspection showed:

- No constant features in the training set.
- No constant features in the test set.

Therefore, all features are retained for further analysis.

## 3.3 Feature Correlation and P-Value Analysis

Feature selection plays a critical role in improving model performance, reducing overfitting, and accelerating the training process. In this project, feature selection was performed exclusively based on the training data to avoid data leakage. The following detailed methodology was adopted:

**Computation of Pearson Correlation and p-Value** For each feature $x_i$ and the target variable $y$, we computed:

- The Pearson correlation coefficient $\rho(x_i, y)$
- The corresponding p-value from the hypothesis test

The Pearson correlation coefficient measures the linear association between two variables and is mathematically defined as:

$$\rho(x_i, y) = \frac{\text{Cov}(x_i, y)}{\sigma_{x_i}\sigma_y}$$

where $\text{Cov}(x_i, y)$ is the covariance between feature $x_i$ and target $y$, and $\sigma_{x_i}$, $\sigma_y$ are their standard deviations respectively.

To assess the statistical significance of the correlation, we conducted a hypothesis test:

$$H_0 : \rho(x_i, y) = 0 \quad \text{(no correlation)}$$
$$H_1 : \rho(x_i, y) \neq 0 \quad \text{(significant correlation)}$$

The p-value quantifies the probability of observing a correlation as extreme as the calculated one under the null hypothesis. A feature was considered statistically significant if its p-value was below the significance level $\alpha = 0.05$.

**Feature Correlation and Significance Results** The table below summarizes the correlation coefficients and p-values between each feature and the target variable:

Table 3.1: Feature correlation and p-value with respect to the target variable.

| Feature | Correlation | P-value |
|---|---|---|
| absoluate_roll | -0.7045 | 0.0000 |
| time1 | 0.6413 | 0.0000 |
| time5 | 0.6404 | 0.0000 |

| Feature | Correlation | P-value |
|---|---|---|
| time2 | 0.6404 | 0.0000 |
| time3 | 0.6404 | 0.0000 |
| time4 | 0.6404 | 0.0000 |
| time7 | 0.6388 | 0.0000 |
| time6 | 0.6388 | 0.0000 |
| time9 | 0.6364 | 0.0000 |
| time8 | 0.6364 | 0.0000 |
| time11 | 0.6337 | 0.0000 |
| time10 | 0.6336 | 0.0000 |
| time12 | 0.6308 | 0.0000 |
| time13 | 0.6308 | 0.0000 |
| time14 | 0.6283 | 0.0000 |
| set | 0.6283 | 0.0000 |
| omega | 0.6159 | 0.0000 |
| n | 0.3970 | $1.61 \times 10^{-309}$ |
| m | 0.3344 | $1.01 \times 10^{-214}$ |
| current_pitch | 0.3081 | $7.00 \times 10^{-181}$ |
| current_roll | 0.1060 | $4.68 \times 10^{-22}$ |
| acc_rate | 0.0796 | $4.35 \times 10^{-13}$ |
| climb_delta_diff | -0.0736 | $2.24 \times 10^{-11}$ |
| track | -0.0613 | $2.53 \times 10^{-8}$ |
| climb_delta | -0.0570 | $2.24 \times 10^{-7}$ |
| roll_rate_delta | -0.0528 | $1.62 \times 10^{-6}$ |
| time9_delta | 0.0402 | $2.60 \times 10^{-4}$ |
| time1_delta | 0.0353 | $1.34 \times 10^{-3}$ |
| time7_delta | 0.0321 | $3.53 \times 10^{-3}$ |
| time11_delta | 0.0273 | $1.30 \times 10^{-2}$ |
| time13_delta | 0.0258 | $1.90 \times 10^{-2}$ |
| time14_delta | -0.0243 | $2.73 \times 10^{-2}$ |
| time4_delta | -0.0194 | $7.76 \times 10^{-2}$ |
| time6_delta | -0.0194 | $7.76 \times 10^{-2}$ |
| time8_delta | -0.0172 | $1.18 \times 10^{-1}$ |
| time3_delta | 0.0166 | $1.31 \times 10^{-1}$ |
| time5_delta | 0.0106 | $3.34 \times 10^{-1}$ |
| time2_delta | -0.0090 | $4.11 \times 10^{-1}$ |
| time12_delta | -0.0089 | $4.18 \times 10^{-1}$ |
| time10_delta | -0.0029 | $7.94 \times 10^{-1}$ |

**Feature Selection Strategy** Based on the above results:

- Features with p-value greater than 0.05 were dropped. These include: `time2_delta`, `time3_delta`, `time4_delta`, `time5_delta`, `time6_delta`, `time8_delta`, `time10_delta`, and `time12_delta`.
- Subsequently, redundant features with strong linear dependence (zero p-value between features) were analyzed and features having 10 or more such dependencies were removed, except for `time1` which was retained manually.

The final selected feature set was saved for consistent application across training and testing datasets.

### 3.4 Feature Scaling

Feature scaling is a crucial preprocessing step to normalize the range of independent variables, ensuring that the learning algorithm treats all features equally and converges faster.

In our preprocessing pipeline, different types of scaling techniques are applied to different groups of features based on their characteristics:

- **Standard Scaling** is applied to features with approximately Gaussian distributions.
- **Robust Scaling** is used for features that may contain outliers.
- **Min-Max Scaling** is used for features where preserving the exact distribution shape and bounds is important.

Specifically:

- Standard scaling features: `acc_rate`, `track`, `current_roll`, `climb_delta`, `roll_rate_delta`, `climb_delta_diff`.
- Robust scaling features: `m`, `absoluate_roll`, `time1`.
- Min-max scaling features: `time1_delta`, `time7_delta`, `time9_delta`, `time11_delta`, `time13_delta`, `time14_delta`.

We use a `ColumnTransformer` to simultaneously apply different scaling strategies to different subsets of features. The fitted transformer is then used to scale both the training and testing data.

The scaling techniques are formally defined as follows:

**Standard Scaling** Standard scaling transforms a feature $x$ according to:

$$x' = \frac{x - \mu}{\sigma}$$

where $\mu$ is the mean of the feature and $\sigma$ is the standard deviation.

**Robust Scaling** Robust scaling transforms a feature $x$ according to:

$$x' = \frac{x - \text{Median}(x)}{\text{IQR}(x)}$$

where $\text{IQR}(x)$ denotes the interquartile range (the difference between the 75th and 25th percentiles).

**Min-Max Scaling** Min-max scaling transforms a feature $x$ into a given range $[a, b]$ (in this case, $[-1, 1]$):

$$x' = a + (b - a) \times \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

where $x_{\min}$ and $x_{\max}$ are the minimum and maximum values of the feature respectively.

After scaling, the features are concatenated back together, maintaining their original relative ordering. The fitted transformer is saved for future inference to ensure consistent transformation across both training and unseen data.

### 3.5 Processed Dataset Summary

After the feature selection and preprocessing steps, the final datasets used for model training and evaluation are summarized as follows.

**Final Dataset Shapes** The final training set contains 8250 samples and 15 features, while the final test set contains 5500 samples and 15 features.

**Selected Features** The 15 features retained after selection are:

`acc_rate`, `track`, `current_roll`, `climb_delta`, `roll_rate_delta`,
`climb_delta_diff`, `m`, `absoluate_roll`, `time1`, `time1_delta`,
`time7_delta`, `time9_delta`, `time11_delta`, `time13_delta`, `time14_delta`

**Feature Selection Summary** Initially, there were 40 features. After feature selection, 15 features were retained, and 25 features were removed. Thus, approximately 62.5% of the original features were eliminated, retaining only the most relevant ones for model development. This dimensionality reduction is expected to improve both model performance and computational efficiency.

# 4 Neural Network and Training Strategies

In this section, we present the methodology behind selecting a neural network (NN) model for the regression task, the hyperparameter tuning approach using random search, and the advanced techniques incorporated into the network architecture to enhance generalization. We also describe the final training procedure and prediction generation.

## 4.1 Model Selection Rationale

Given the regression nature of the task and the structure of the provided dataset, we selected a deep feed-forward neural network (Multilayer Perceptron, MLP) as our primary modeling framework. This decision was informed by empirical results from Milestones 1 and 2, where neural networks consistently outperformed baseline models such as Ridge Regression and Random Forests in terms of model performance.

Compared to linear models, MLPs have the following advantages that motivated our model choice:

- **Nonlinear modeling capabilities**: Neural networks can capture complex, nonlinear relationships between input features and target values, making them well-suited for structured data regression tasks.
- **Flexibility and scalability**: By adjusting the depth and width of the network, we can easily control model capacity and adapt to different levels of data complexity.
- **Robustness with regularization**: Techniques like dropout, weight decay, and batch normalization can be seamlessly integrated to prevent overfitting and stabilize training.

Although other model families such as gradient boosting machines (e.g., XGBoost, LightGBM) are often competitive for structured data, we prioritized neural networks in this project to explore deep learning-based strategies and to fully utilize architectural regularization techniques.

A detailed description of the final network architecture is provided in Section 4.2.

## 4.2 Model Architecture

The final network architecture consists of an input layer, three hidden layers, and an output layer. All layers are fully connected. Batch normalization and dropout are applied after each of the first two hidden layers to improve convergence and generalization.

- **Input Layer**: 15 units (equal to the number of features).
- **Hidden Layer 1**: 64 units, followed by BatchNorm, ReLU, and Dropout (rate = 0.2).
- **Hidden Layer 2**: 32 units, followed by BatchNorm, ReLU, and Dropout (rate = 0.2).
- **Hidden Layer 3**: 16 units, followed by BatchNorm and ReLU.
- **Output Layer**: 1 unit (regression target).

## 4.3 Neural Network Techniques

Several techniques were incorporated into the neural network model to enhance its generalization ability, stability, and training efficiency.

**1. $L_2$ Regularization (Weight Decay).** Weight decay is applied through the optimizer to penalize large weight magnitudes, thereby encouraging simpler and more generalizable models. The weight decay coefficient $\lambda$ was treated as a tunable hyperparameter during random search, sampled from a log-uniform distribution.

**2. Dropout Regularization.** Dropout layers with a drop probability of 0.2 are added after each hidden layer. During training, neurons are randomly deactivated, forcing the network to learn redundant and robust representations, thus reducing overfitting.

**3. Batch Normalization.** Batch Normalization (BN) layers are inserted before each ReLU activation function. BN normalizes the inputs to each layer, stabilizing the learning process, enabling the use of higher learning rates, and accelerating convergence.

**4. Early Stopping.** Early stopping is employed with a patience of 30 epochs. If the validation RMSE does not improve for 30 consecutive epochs during training, the training process is terminated early to prevent overfitting and unnecessary computation.

**5. Ensemble Learning via K-Fold Cross-Validation.** A 5-fold cross-validation (CV) strategy is used during both hyperparameter tuning and final model training. For the final prediction, models trained on each fold are ensembled by averaging their predictions, resulting in more robust and stable outputs.

## 4.4 Random Search for Hyperparameter Tuning

Instead of conducting an exhaustive grid search, we adopt a **Random Search** approach, which is more efficient for exploring high-dimensional hyperparameter spaces.

The hyperparameters tuned are:

$$\begin{aligned} \text{Learning rate} \quad & \alpha \sim \log_{10} Uni(-4, -2) \\ \text{Batch size} \quad & n \in \{8, 16, 32, 64\} \\ \text{Weight decay} \quad & \lambda \sim \log_{10} Uni(-5, -3) \end{aligned}$$

where $Uni(a, b)$ denotes a uniform distribution between $10^a$ and $10^b$ on a logarithmic scale.

The random search procedure is as follows:

1. Randomly sample a combination of hyperparameters.
2. For the sampled configuration, perform a 5-fold cross-validation.
3. For each fold:
   - Train the model with early stopping based on validation RMSE.
   - Record the best validation RMSE achieved within the patience window.
4. Compute the average validation RMSE across the five folds.

A total of 5 random search trials are conducted. The hyperparameter set yielding the lowest average validation RMSE is selected as the final configuration.

During each trial, the training, validation, and evaluation processes are carried out independently, and the random search results (hyperparameters and their corresponding average validation RMSE) are stored for future analysis.

## 4.5 Results of Random Search

After conducting random search, the relationship between hyperparameters and model performance is analyzed by observing how the validation RMSE varies with:

- Learning rate
- Weight decay
- Batch size

The analysis shows that the learning rate and regularization strength (weight decay) significantly affect model performance. Specifically, an appropriate learning rate ensures convergence without overshooting, while a well-tuned weight decay prevents overfitting. The batch size exhibited a relatively minor influence, although smaller batch sizes sometimes led to slightly better performance due to the regularizing effect of noisier gradients.

## 4.6 Best Hyperparameters and Final Model Training

The best hyperparameters found by random search are:

- Learning rate: $\alpha =$ (insert best value here)
- Batch size: $n =$ (insert best value here)
- Weight decay: $\lambda =$ (insert best value here)

With the best hyperparameters selected:

- A new 5-fold cross-validation is conducted on the full training set.
- In each fold, early stopping is applied to select the best model.
- After training, each fold's model is saved and used to predict the test set.

The final test predictions are obtained by averaging the outputs from the five trained models:

$$\hat{y}_{\text{final}} = \frac{1}{5} \sum_{k=1}^{5} \hat{y}_{\text{fold } k}$$

where $\hat{y}_{\text{fold } k}$ denotes the prediction from the model trained on the $k$-th fold.
The ensembled predictions are saved in `submission.csv` for final submission.

## 4.7 Performance of Neural Network

After completing the hyperparameter tuning and model training procedures, we evaluated the performance of the final neural network model as follows.

**Random Search Trials** The results of each random search trial, including sampled hyperparameters and their corresponding validation RMSE, are summarized in Table 4.1.

| Trial | Learning Rate | Batch Size | Weight Decay | Avg Validation RMSE |
|---|---|---|---|---|
| 1 | $1.33 \times 10^{-3}$ | 16 | $3.00 \times 10^{-5}$ | 0.16220 |
| 2 | $7.71 \times 10^{-3}$ | 32 | $5.00 \times 10^{-5}$ | 0.16310 |
| 3 | $1.60 \times 10^{-4}$ | 16 | $1.10 \times 10^{-4}$ | 0.16522 |
| 4 | $2.87 \times 10^{-3}$ | 8 | $4.60 \times 10^{-4}$ | 0.17295 |
| 5 | $5.07 \times 10^{-3}$ | 64 | $2.05 \times 10^{-5}$ | **0.16083** |

Table 4.1: Summary of Random Search Trials and Corresponding Validation RMSE.

**Best Hyperparameters Found** Through random search and 5-fold cross-validation, the best hyperparameters identified were:

$$\text{Learning rate} \quad \alpha = 5.07 \times 10^{-3}$$
$$\text{Batch size} \quad n = 64$$
$$\text{Weight decay} \quad \lambda = 2.05 \times 10^{-5}$$

The corresponding average validation RMSE achieved during cross-validation was:

$$\text{Best Validation RMSE} = 0.16083$$

**Final Model Training and Submission** Using the best hyperparameters, we retrained the neural network with a 5-fold cross-validation strategy. In each fold, the best model (selected based on validation RMSE with early stopping) was saved. The final test set predictions were generated by averaging the outputs of the five trained models:

$$\hat{y}_{\text{final}} = \frac{1}{5} \sum_{k=1}^{5} \hat{y}_{\text{fold } k}$$

The ensembled predictions were saved to `submission.csv` and submitted to the Kaggle competition.

**Kaggle Submission Results** The final submission achieved a test RMSE score of:

$$\text{Kaggle RMSE} = 0.16295$$

This result outperformed both provided baselines:

- Baseline 1: RMSE = 0.20000
- Baseline 2: RMSE = 0.17000

Thus, the proposed neural network model significantly improved over the baseline models, demonstrating the effectiveness of systematic hyperparameter tuning and careful model regularization.

# 5 Conclusion

## 5.1 Summary of Work

This project focused on developing a regression model for a classroom Kaggle competition, where performance was evaluated based on Root Mean Squared Error (RMSE) on a private test set. We systematically addressed the entire machine learning pipeline from data preprocessing to final model ensembling.

We began with thorough data cleaning, feature selection using Pearson correlation and p-values, and custom scaling for heterogeneous feature groups using a ColumnTransformer. Our final model was a feedforward artificial neural network (ANN) implemented in PyTorch, incorporating Batch Normalization, Dropout, and L2 regularization to improve generalization and stability. Hyperparameter tuning was conducted via random search across key parameters including learning rate, batch size, and weight decay. Final predictions were obtained by ensembling results from five models trained through 5-fold cross-validation to enhance stability and robustness.

## 5.2 Main Results

The final model was trained using the best hyperparameter configuration selected via random search and validated with 5-fold cross-validation. Test predictions were generated by averaging outputs from five models trained on separate folds.

- The best average validation RMSE achieved during cross-validation was **0.16083**.

- The ensemble achieved a Root Mean Squared Error (RMSE) of **0.16295** on the public Kaggle leaderboard.

## 5.3 Limitations

While the final neural network model achieved solid performance in the Kaggle competition, several limitations were observed in our approach:

- **Limited Model Exploration:** We focused exclusively on a feedforward neural network (ANN) architecture. Other regression models such as gradient boosting methods (e.g., XGBoost, LightGBM) or ensemble techniques were not explored, which might potentially achieve better performance on structured tabular data.
- **Basic Feature Selection:** Feature selection was based solely on linear correlation and p-value analysis. This method may overlook nonlinear relationships between features and the target variable, leading to potential loss of useful information.
- **Small Random Search Space:** Only 5 random search trials were conducted for hyperparameter tuning due to computational constraints. A larger search space or more sophisticated tuning strategies (such as Bayesian Optimization) might yield better hyperparameters.
- **Absence of Extensive Regularization Techniques:** Although weight decay and dropout were used, other advanced regularization methods (such as label smoothing, data augmentation for structured data, or mixup techniques) were not considered.
- **Potential Overfitting to Validation Folds:** Despite using early stopping, there remains a possibility of slight overfitting to the validation folds, as model selection was based on performance on validation sets without a fully held-out unseen validation set.

## 5.4 Future Works

Building upon the findings and limitations of this project, several directions can be considered for future work:

- **Model Diversification:** Investigate alternative machine learning models tailored for tabular data, such as Gradient Boosted Decision Trees (GBDT), CatBoost, or TabNet, and compare their performance with neural networks.
- **Advanced Feature Engineering:** Employ more sophisticated feature selection techniques, including mutual information analysis, recursive feature elimination (RFE), and tree-based feature importance methods, to capture non-linear interactions.
- **Enhanced Hyperparameter Optimization:** Apply more powerful hyperparameter tuning methods such as Bayesian Optimization (e.g., with Optuna or Hyperopt) to explore the hyperparameter space more efficiently and thoroughly.
- **Ensemble Methods:** Combine predictions from multiple different model types (e.g., ANN + GBDT) using stacking or blending techniques to potentially boost overall performance.
- **Cross-Validation Strategy Improvements:** Implement nested cross-validation to better estimate generalization performance and reduce bias from overfitting of hyperparameter selection.
- **Data Augmentation for Structured Data:** Explore data augmentation strategies specifically designed for structured datasets, such as SMOTE, Mixup for tabular data or feature perturbations.