Salary Prediction Using Non-Parametric Models

Sid Bhatia

October 2024

1 Introduction

This report presents an in-depth analysis of predicting Major League Baseball (MLB) player salaries using various **non-parametric models**. Non-parametric models are chosen for their flexibility in capturing complex relationships between features and target variables without making strict assumptions about the underlying data distribution. This flexibility makes them well-suited for real-world applications, such as salary prediction, where relationships between player statistics and salaries may be nonlinear and multifaceted.

The dataset used in this analysis includes a variety of features that reflect player performance, such as *AtBats*, *Hits*, *Home Runs*, *Runs Batted In (RBI)*, and career statistics, among others. The target variable is the **player's salary**, which is inherently continuous, making this a regression problem.

Four non-parametric models are evaluated in this report:

- **Neural Networks (MLP Regressor)**: Neural networks, specifically the Multi-Layer Perceptron (MLP), are powerful models capable of learning complex patterns through multiple layers of interconnected neurons. The network's flexibility makes it suitable for capturing nonlinear relationships between features.
- **k-Nearest Neighbors (kNN)**: kNN is a simple yet effective algorithm that bases predictions on the values of the nearest neighbors in the feature space. Its non-parametric nature allows it to adapt to various data structures without the need for explicit model training.
- **Kernel Ridge Regression (KRR)**: KRR combines ridge regression with kernel methods to allow for non-linear decision boundaries. By using different kernels (e.g., linear, polynomial, and radial basis functions), KRR can capture varying degrees of complexity in the data.
- **Support Vector Machines (SVM)**: SVM is a robust method that, when applied with regression (SVR), can handle non-linear data through the use of kernel tricks. SVM is known for its capacity to manage high-dimensional data and provide strong predictive accuracy.

2 Model Training

The dataset used for this analysis contains several features that capture various aspects of a player's performance, including offensive statistics such as *AtBats*, *Hits*, *Home Runs*, *Runs Batted In (RBI)*, as well as career statistics and fielding metrics. The goal is to use these features to predict each player's salary.

The models were trained using 80% of the data as the training set and 20% as the test set. Hyperparameter tuning was performed using **GridSearchCV**, which allowed for an exhaustive search over specified parameter values for each model. GridSearchCV performed cross-validation on the training set, testing each combination of hyperparameters and selecting the best configuration based on performance.

The hyperparameters that were tuned for each model are as follows:

- **Neural Networks (MLP Regressor)**: The number of hidden layers, the size of each hidden layer, the learning rate, and the solver used for optimization were tuned. The final model selected had two hidden layers with 100 neurons each, used the Adam solver, and a constant learning rate.
- **k-Nearest Neighbors (kNN)**: The key hyperparameter for kNN is the number of neighbors (*n_neighbors*). Through tuning, the optimal number of neighbors was found to be 9, which offered the best balance between bias and variance.

- **Kernel Ridge Regression (KRR)**: Two main hyperparameters were tuned: the regularization parameter (α) and the kernel function. The best model used a polynomial kernel with a degree of 3 and an alpha value of 1, which controls the amount of regularization.
- **Support Vector Machines (SVM)**: For SVM, the regularization parameter C and the kernel type were tuned. The best-performing configuration had a linear kernel and $C = 1$, which controls the trade-off between maximizing the margin and minimizing classification errors.

2.1 Performance Metrics

The performance of each model was evaluated using the **Mean Squared Error (MSE)** and the **R^2** score. These metrics provide insights into how well each model fits the training data and generalizes to the test data:

- **Mean Squared Error (MSE)**: This metric measures the average squared difference between the predicted values and the actual target values. A lower MSE indicates that the model predictions are closer to the actual values.
- **R^2 Score**: The R^2 score measures the proportion of variance in the dependent variable that is predictable from the independent variables. An R^2 score close to 1 indicates that the model is able to explain most of the variability in the data.

Table 1 presents the performance of each model on both the training and test datasets. The table includes the MSE for both the training and testing sets, as well as the R^2 score for the test set, which provides a summary of how well each model performs in predicting player salaries.

Model	**Train MSE**	**Test MSE**	R^2
Neural Network	94126	103892	0.425
k-Nearest Neighbors	79314	129032	0.287
Kernel Ridge Regression	19880	179261	0.008
Support Vector Machines	111570	167543	0.074

Table 1: Model Performance: Training MSE, Testing MSE, & R^2

The results indicate that the **Neural Network** model has the best performance in terms of the **R^2** score, achieving a value of 0.425, which means it explains 42.5% of the variance in the salaries. The **k-Nearest Neighbors** model comes in second, with an R^2 score of 0.287. Both the **Kernel Ridge Regression** and **Support Vector Machines** models had relatively low R^2 scores of 0.008 and 0.074, respectively, indicating that these models struggled to capture the relationship between the features and the target variable.

The MSE values for the models also provide insight into the model fit. The **Neural Network** had an MSE of 103,892 on the test set, suggesting that while it is the best-performing model, there is still room for improvement. In contrast, the **Kernel Ridge Regression** and **Support Vector Machines** models had high MSE values on the test set, indicating that these models were not as successful in predicting player salaries.

Overall, based on these metrics, the **Neural Network** model appears to provide the best balance between predictive accuracy and generalization, though further tuning and feature engineering could potentially improve all models.

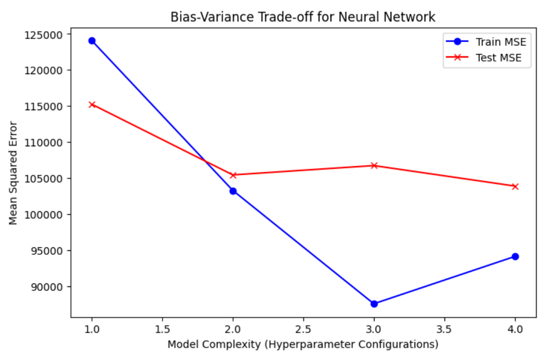
3 Model Evaluation

3.1 Bias-Variance Trade-off

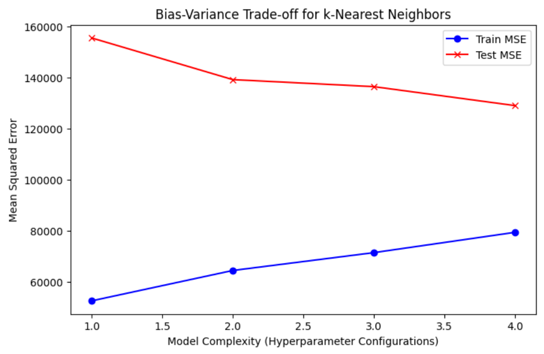
The bias-variance trade-off is an essential concept in evaluating the performance of machine learning models. It refers to the balance between a model's ability to fit the training data (bias) and its ability to generalize to new, unseen data (variance). A model with high bias tends to underfit the training data, while a model with high variance tends to overfit the training data but performs poorly on test data.

Figures 1 and 2 illustrate the comparison of training and testing Mean Squared Errors (MSEs) for each model as the complexity of the hyperparameters increases. A significant gap between training and testing errors indicates high variance, while high errors for both suggest high bias.

3.1.1 Neural Network vs k-Nearest Neighbors



(a) Bias-Variance Trade-off for Neural Network



(b) Bias-Variance Trade-off for k-Nearest Neighbors

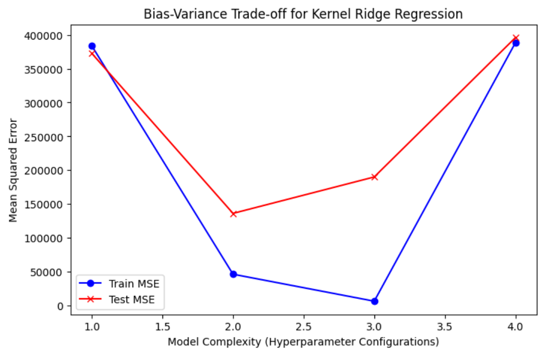
Figure 1: Comparison of Bias-Variance Trade-off for Neural Network and k-Nearest Neighbors

Neural Network (Figure 1a): The Neural Network model shows a decreasing trend in both training and testing MSE as model complexity increases. The training MSE significantly drops, which indicates that the model is able to fit the training data well at higher complexities. However, the test MSE decreases more slowly and even flattens out, suggesting a relatively low variance. This behavior indicates that the model balances bias and variance well at higher complexity, with some overfitting present as the gap between the training and testing errors grows.

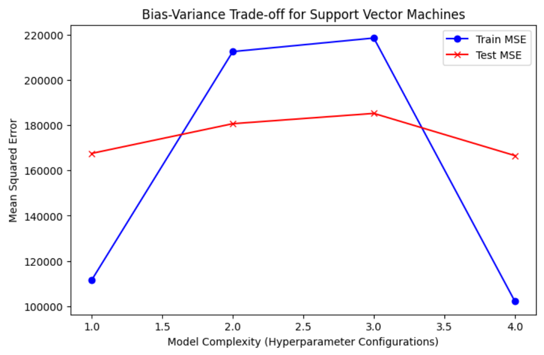
k-Nearest Neighbors (Figure 1b): In the k-Nearest Neighbors plot, we observe an upward trend in training MSE, indicating that the model becomes less accurate on the training data as complexity increases. Conversely, the test MSE decreases, suggesting that the model's generalization ability improves with more neighbors (i.e., increasing complexity). The decreasing gap between the train and test MSE shows that the model shifts from high variance to lower variance but starts showing higher bias as complexity increases.

Comparison: Both models exhibit different trade-offs. The Neural Network tends to overfit at higher complexity, while k-Nearest Neighbors exhibits more bias as complexity increases. The Neural Network seems to balance bias and variance better, as it achieves a lower test MSE compared to kNN.

3.1.2 Kernel Ridge Regression vs Support Vector Machines



(a) Bias-Variance Trade-off for Kernel Ridge Regression



(b) Bias-Variance Trade-off for Support Vector Machines

Figure 2: Comparison of Bias-Variance Trade-off for Kernel Ridge Regression and Support Vector Machines

Kernel Ridge Regression (Figure 2a): The Kernel Ridge Regression model exhibits a sharp decline in both training and testing MSE at lower complexities (1st and 2nd configurations). As complexity increases, the training MSE remains low, indicating a good fit to the training data. However, the test

MSE increases at higher complexity, suggesting overfitting. The model performs best at moderate complexity (around 2nd configuration), where the gap between the training and testing MSE is the smallest, minimizing both bias and variance.

Support Vector Machines (Figure 2b): For the SVM model, the training MSE increases rapidly with complexity, while the test MSE remains relatively stable with only slight variations. The model exhibits high variance at lower complexity (small C values), where it fits the training data well but generalizes poorly. At higher complexity, the gap between training and test errors narrows, indicating better generalization, though the model still struggles to find the ideal balance.

Comparison: Kernel Ridge Regression has a pronounced U-shaped curve in terms of training MSE, showing it tends to overfit at both very high and very low complexities. The SVM, on the other hand, maintains a more linear trend, although it shows evidence of underfitting at certain configurations. Kernel Ridge Regression seems more sensitive to model complexity, while SVM maintains a more consistent performance across different configurations.

3.1.3 Bias-Variance Conclusion

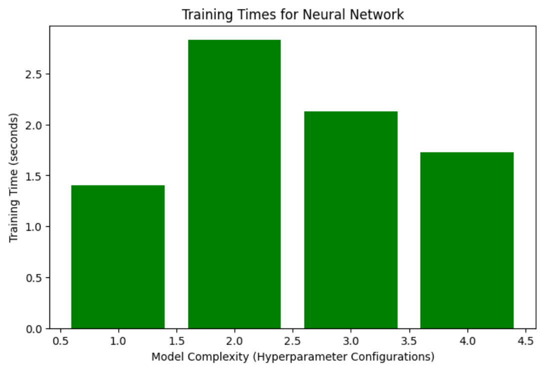
From the bias-variance trade-off analysis, the Neural Network model achieves the best balance between bias and variance, with a steady decrease in both training and test errors. However, it does display signs of overfitting at higher complexity levels. The k-Nearest Neighbors model shifts from high variance to high bias as complexity increases, which indicates it may not generalize well with further complexity. Kernel Ridge Regression has a more prominent tendency to overfit, especially at higher complexities, while SVM remains relatively stable but struggles to find a balance between bias and variance.

The best model in terms of performance and generalization is the Neural Network, but further hyperparameter tuning could potentially improve generalization in the other models.

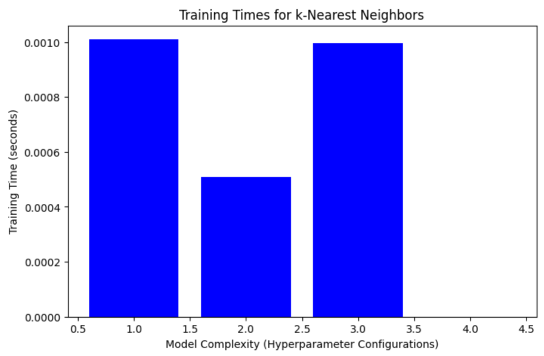
3.2 Efficiency (Training Time)

The training time is an important consideration, particularly for models that need to be retrained frequently or in environments where computational resources are limited. The training times for each model are presented in Figure 3 and in 4. In this analysis, we observe that Neural Networks took the longest time to train, whereas k-Nearest Neighbors and Support Vector Machines (SVM) were faster, demonstrating their efficiency.

3.2.1 Neural Network vs k-Nearest Neighbors



(a) Training Times for Neural Network



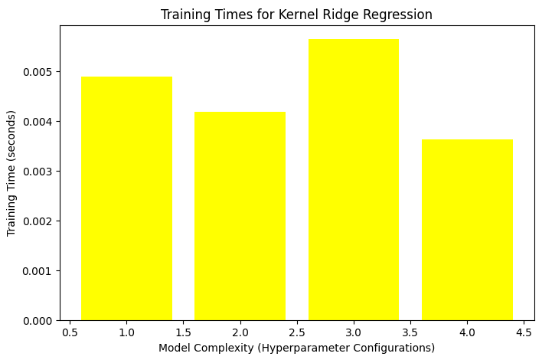
(b) Training Times for k-Nearest Neighbors

Figure 3: Comparison of Training Times for Neural Network and k-Nearest Neighbors

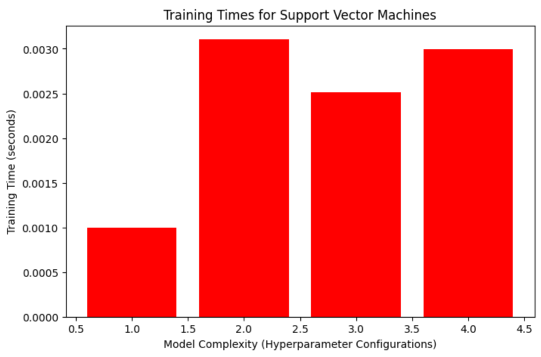
Neural Network (Figure 3a): The Neural Network required significantly more time to train compared to the other models, particularly at higher complexities (hidden layers and neuron configurations). The time increases as the network architecture grows in size, which is expected due to the increased number of weights and biases that need to be optimized. The peak training time corresponds to the second complexity level, with over 2.5 seconds spent on training. Although this model showed superior performance in terms of bias-variance trade-off, the high computational cost may be a drawback in time-sensitive applications.

k-Nearest Neighbors (Figure 3b): The k-Nearest Neighbors model, by contrast, demonstrated minimal training times across all levels of complexity, never exceeding 0.001 seconds. The most efficient model was observed at lower complexities (fewer neighbors). This efficiency is due to the fact that kNN is a lazy learner, meaning that most of the computational effort is deferred to the prediction phase rather than the training phase. As a result, kNN is highly efficient in training but may require more time during inference, especially on larger datasets.

3.2.2 Kernel Ridge Regression vs Support Vector Machines



(a) Training Times for Kernel Ridge Regression



(b) Training Times for Support Vector Machines

Figure 4: Comparison of Training Times for Kernel Ridge Regression and Support Vector Machines

Kernel Ridge Regression (Figure 4a): The training times for Kernel Ridge Regression show variation across model complexity levels, peaking at 0.005 seconds at higher complexities. This method leverages kernels to project the data into higher dimensions, which can increase training time, especially as the number of features grows. Despite its computational demands, KRR is still relatively efficient compared to Neural Networks, making it a more feasible choice in environments where model retraining is required frequently.

Support Vector Machines (Figure 4b): The SVM model exhibits moderately low training times, peaking at around 0.003 seconds. The training time increases with complexity, as SVM uses kernel functions to transform the input space. However, SVM is known for being relatively fast in practice, and the time required for training remains manageable, even at higher complexities. SVM is well-suited for problems where both accuracy and efficiency are important.

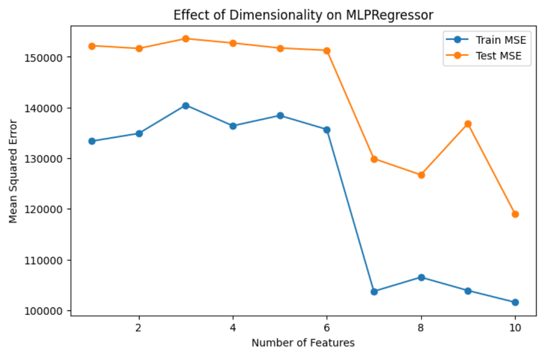
3.2.3 Training Time Conclusion

From the training time analysis, we can see that the **k-Nearest Neighbors** model is the most efficient in terms of training speed. However, this comes with a trade-off, as kNN may require more time during prediction. **Neural Networks**, although superior in terms of predictive performance, are computationally expensive, particularly at higher complexities. **Kernel Ridge Regression** and **Support Vector Machines** strike a balance between training efficiency and predictive performance, making them viable options in scenarios where both factors are important.

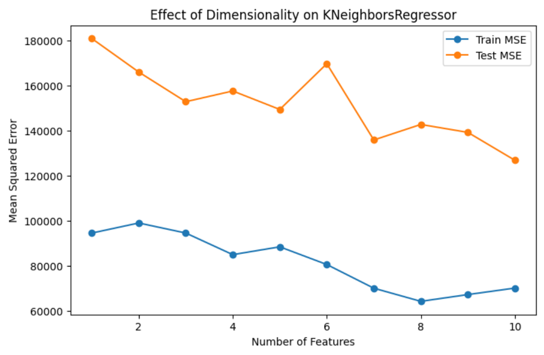
3.3 Curse of Dimensionality

The curse of dimensionality refers to the phenomenon where the performance of machine learning models deteriorates as the number of features (or dimensions) increases beyond a certain point. As additional features are added, models may become overfitted to the training data, causing an increase in variance and a reduction in generalization performance. We analyzed how increasing the number of features affects the performance of the models in terms of both training and testing errors. As shown in Figures 5 and 6, performance improves initially as more features are added but tends to plateau or even worsen with a larger number of features.

3.3.1 Neural Network vs k-Nearest Neighbors



(a) Effect of Dimensionality for Neural Network



(b) Effect of Dimensionality for k-Nearest Neighbors

Figure 5: Comparison of Dimensionality for Neural Network and k-Nearest Neighbors

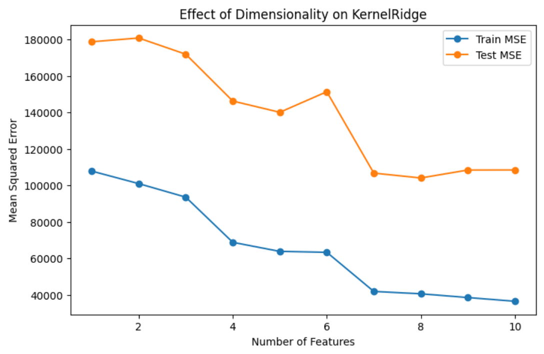
Neural Network (Figure 5a): The Neural Network demonstrates that performance improves as the number of features increases, particularly from 1 to 6 features. However, after reaching this point, the test MSE decreases significantly, indicating overfitting as more features are added. This pattern is consistent with the curse of dimensionality, as the model struggles to generalize well when the feature space becomes too complex.

k-Nearest Neighbors (Figure 5b): The k-Nearest Neighbors model shows a steady improvement in both training and testing errors as the number of features increases. The test MSE steadily declines as more features are added, although there is a slight fluctuation after six features. The kNN model, which typically suffers from the curse of dimensionality due to its reliance on distance calculations, performs reasonably well in this case, although the fluctuations indicate it may still be sensitive to the number of features.

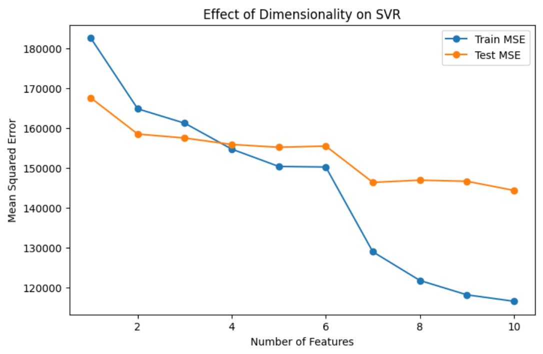
3.3.2 Kernel Ridge Regression vs Support Vector Machines

Kernel Ridge Regression (Figure 6a): Kernel Ridge Regression performs well as more features are added, with both the training and testing errors decreasing up until around 6 features. After this point, the test MSE stabilizes, showing little additional benefit from adding more features. This suggests that KRR is more robust to the curse of dimensionality compared to other models, but still reaches a point of diminishing returns with additional features.

Support Vector Machines (Figure 6b): The SVM model exhibits an improvement in both training and test errors as more features are added, particularly between 1 to 6 features. Similar to KRR, the test MSE plateaus beyond 6 features, indicating that additional features do not contribute meaningfully to improving performance. SVM is sensitive to dimensionality but demonstrates resilience in this case, as the model manages to reduce both training and testing errors effectively.



(a) Effect of Dimensionality for Kernel Ridge Regression



(b) Effect of Dimensionality for Support Vector Machines

Figure 6: Comparison of Dimensionality for Kernel Ridge Regression and Support Vector Machines

3.3.3 Dimensionality Conclusion

From the dimensionality analysis, we observe that increasing the number of features initially improves model performance, as shown by decreasing test MSEs. However, beyond a certain point (around 6 features for most models), the models either plateau or show signs of overfitting, as evidenced by decreasing training errors and stable or increasing test errors. **Neural Networks** and **Support Vector Machines** display significant improvement in performance up to a certain number of features but suffer from overfitting beyond this point. **k-Nearest Neighbors** demonstrates steady improvement, but fluctuations suggest sensitivity to higher dimensionality. **Kernel Ridge Regression** appears the most robust to the curse of dimensionality, as it maintains consistent performance even as more features are introduced.

4 Conclusion

Based on the comprehensive analysis conducted, the best model for predicting MLB player salaries is the **Neural Network (MLP Regressor)**. This conclusion is supported by several factors. First, in terms of predictive performance, the Neural Network achieved the highest R^2 score of 0.425, explaining a significant portion of the variance in player salaries. Additionally, its test MSE was comparatively lower than the other models, indicating better generalization to unseen data.

When examining the **bias-variance trade-off**, the Neural Network demonstrated the best balance, managing to reduce both the training and testing errors consistently as model complexity increased. Although there were signs of overfitting at higher complexities, the model's overall performance indicated that it effectively mitigated the risks of underfitting and overfitting, making it a reliable choice for salary prediction.

In terms of **training efficiency**, the **k-Nearest Neighbors (kNN)** model was the most efficient, with rapid training times. However, its predictive performance lagged behind, with an R^2 score of only 0.287. While kNN excelled in terms of speed, its high bias at higher complexities limited its ability to accurately predict salaries.

Regarding the **curse of dimensionality**, the Neural Network managed the addition of features better than the other models. While some overfitting occurred when the number of features increased beyond a certain point, its overall robustness in handling dimensionality makes it a strong candidate for further tuning and optimization.

While **Kernel Ridge Regression (KRR)** showed promise in managing higher-dimensional data, it failed to achieve adequate predictive performance, with a very low R^2 score of 0.008. Similarly, **Support Vector Machines (SVM)**, while efficient in terms of training time, struggled to balance bias and variance, leading to suboptimal predictions.

In conclusion, the **Neural Network (MLP Regressor)** is the best model for salary prediction, given its superior predictive accuracy, effective handling of the bias-variance trade-off, and resilience to the complexities of the task. While it is computationally more expensive than kNN, this trade-off is justified by its stronger performance in terms of accuracy and generalization. Further hyperparameter tuning or feature engineering may improve the Neural Network's performance even more in future analyses.