COMP1801 - Machine Learning Coursework Report

Sara Hodaei – 001421104

Word Count:2970

1. Executive Summary

This report focuses on predicting and classifying the lifespan and usability of metal parts to optimize production quality. Gradient Boosting and MLP were used for regression, while SVM with RBF kernel and MLP were applied for classification. The Gradient Boosting Regressor achieved the highest accuracy (R^2 = 0.9508) in predicting part lifespan, while SVM outperformed MLP in classification with a 94.5% accuracy. Gradient Boosting is recommended for precise lifespan predictions, and SVM for reliable usability classification, due to their robustness.

2. Data Exploration

To prepare analysing and manipulating data, python libraries Pandas and Numpy were added (the first code cell in the .ipynb file shows all the libraries used). The dataset was accessed through google.colab.files and files.upload() within Google Colab environment. The function pd.read_csv() structured the data frame for more efficient analysis. To further investigate the dataset, explore and find correlations, libraries Matplotlib.pyplot and Seaborn were added; responsible for creating figures and plots with labels and drawing informative statistical graphics respectively. Even though the data set was initially clean (no null values), the lines df.info and df.describe provided a structural and statistical overview of the set by giving each column’s data type, min, max and mean. With the help of scipy stats, a box plot and a histogram of each numerical column were used to detect outliers and display distribution, as it informs preprocessing and modelling. Normally distributed features align well with many statistical tests and models, enhancing their performance. All columns showed an approximate symmetrical distribution (Skewness nearly 0, meaning there are no considerable issues in data distribution) except for "smalldefect", "largedefects" and "silverdefects", suggesting the presence of potential noise as shown in fig 1and 2.

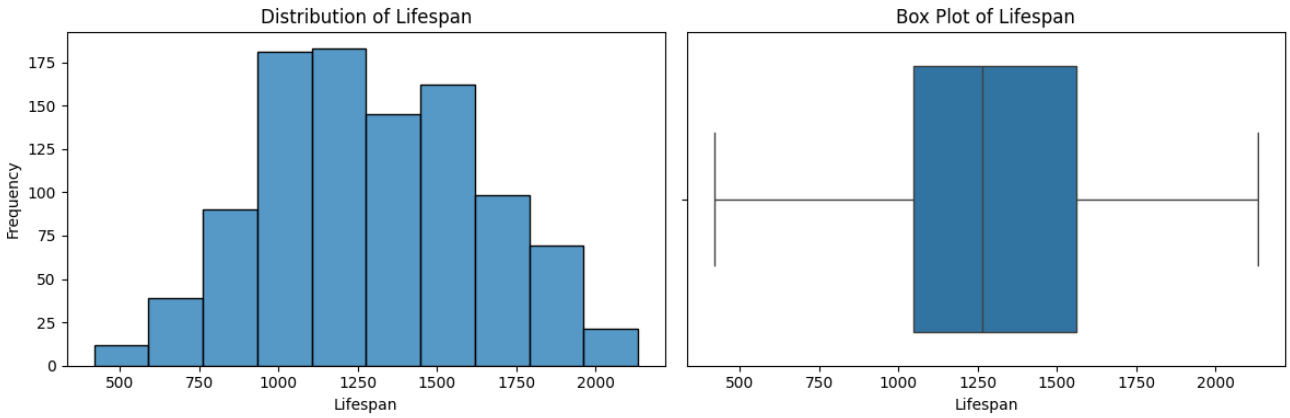

Figure 1: Skewness of Lifespan: 0.077 indicating a normal distribution

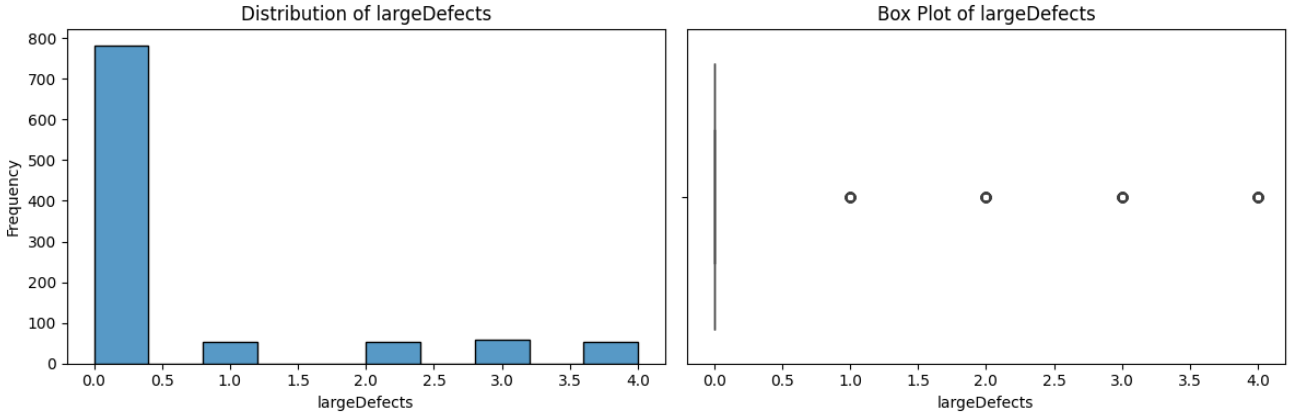

Figure 2: Skewness of "largeDefects": 1.98 showing extreme values

Data preprocessing:

Sklearn.preprocessing import OneHotEncoder, LabelEncoder was used to transform categorical features into numbers to find more effective relationships between features and fit a better model. For the features "partType", "microstructure" and "castType", OHE method was selected as they're nominal features. However, labelling method was chosen for the feature "seedLocation" since it has only two categories and OHE method would only provide two inverted columns of zeros and ones. To avoid redundancy the original categorical features were deleted from the data frame's copy.

The dataset was divided into training, validation, and testing sets. 80% of the data was allocated for training and 20% for testing. The training set was further split, resulting in 60% for training and 20% for validation to prevent overfitting (Haykin, 1999).

Numerical features are scaled to prevent features with wider ranges from dominating the model and to ensure all features contribute equally to the learning process. However, categorical features were left untouched to preserve their natural properties. Data scaling was performed after splitting the data to prevent data leakage and ensure a fair model evaluation. Numerical features were

standardized using StandardScaler. Standardization was chosen over normalization to handle potential outliers in these features and make the model less sensitive to feature scales. Standardization involves centering the data around zero and scaling it to unit variance. Fit_transform was applied to the training data to learn the mean and standard deviation for scaling. Then, transform was used on the validation and test data, utilizing the scaling parameters learned from the training set.

Examining Correlations:

Due to the presence of categorical variables, the ANOVA F-test method was tested solely on training data (for avoiding data leakage) to find relationships, not necessarily correlations, between them and numerical features. A P-value lower than 0.05 suggests considerable relationships among the two features; which was the case for variables "partType_1", "partType_3", "castType_0" and "castType_2" as illustrated in figure 3.

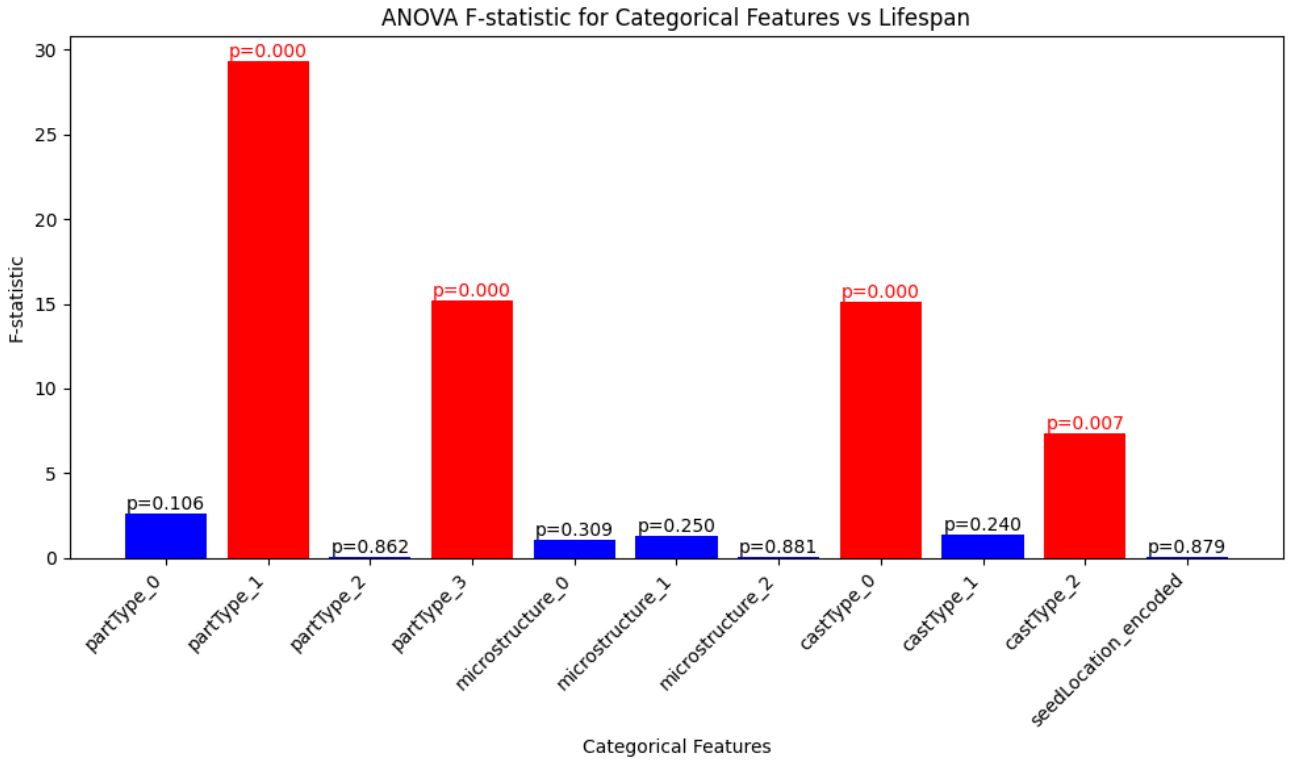

Figure 3: Red bars indicate features with a statistically significant relationship to Lifespan (p-value < 0.05)

The Seaborn library is used to generate a color-coded heatmap that visualizes the correlations between the target variable "Lifespan" and the rest of the features. This heatmap helps to identify features that have strong positive (close to 1), negative (close to -1), or no (close to 0) correlations with the target variable, providing insights for feature selection and model building. These low correlation values found by the Pearson's test in figure 4, suggest that there isn't a strong linear

relationship between the variables. Therefore, more complex models and non-linear relationships were sought. “Nickle” (0.28) and “Iron” (0.21) were the highest positive correlations, however, due to the high negative correlation (-0.79) between these two, Iron was excluded from the feature set to avoid multicollinearity issues. “smalldefects” and “coolingrate” with 0.81 correlation, also add to the matter.

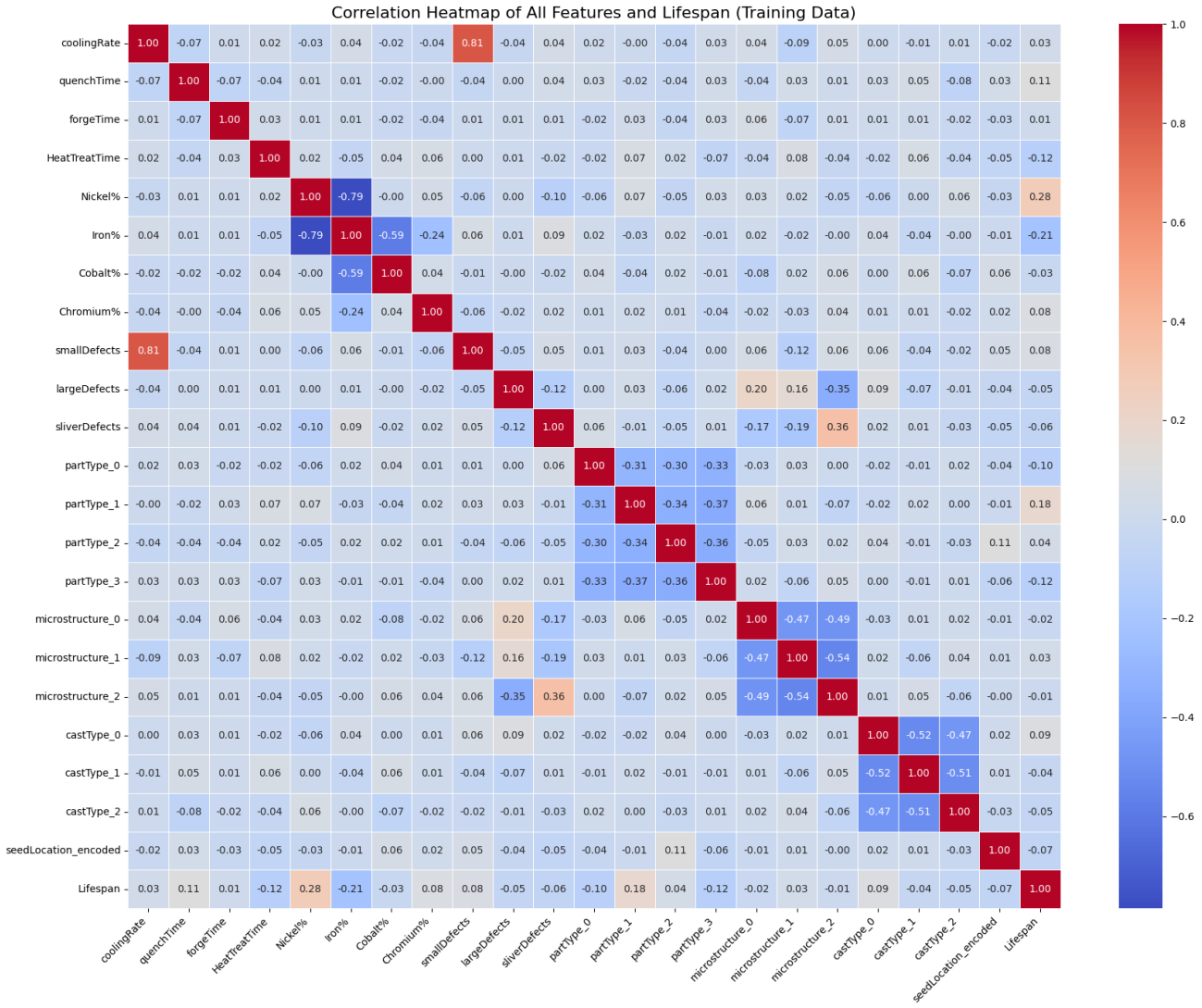

Figure 4

Given the non-linearity shown by the heatmap, the Decision Tree feature importance, fig 5, was used as it can handle non-linearity and it’s immune to outliers and irrelevant features; highlighting “coolingRate”, ”Nickel%”,” HeatTreatTime”,” partType_3” and “partType_1”.

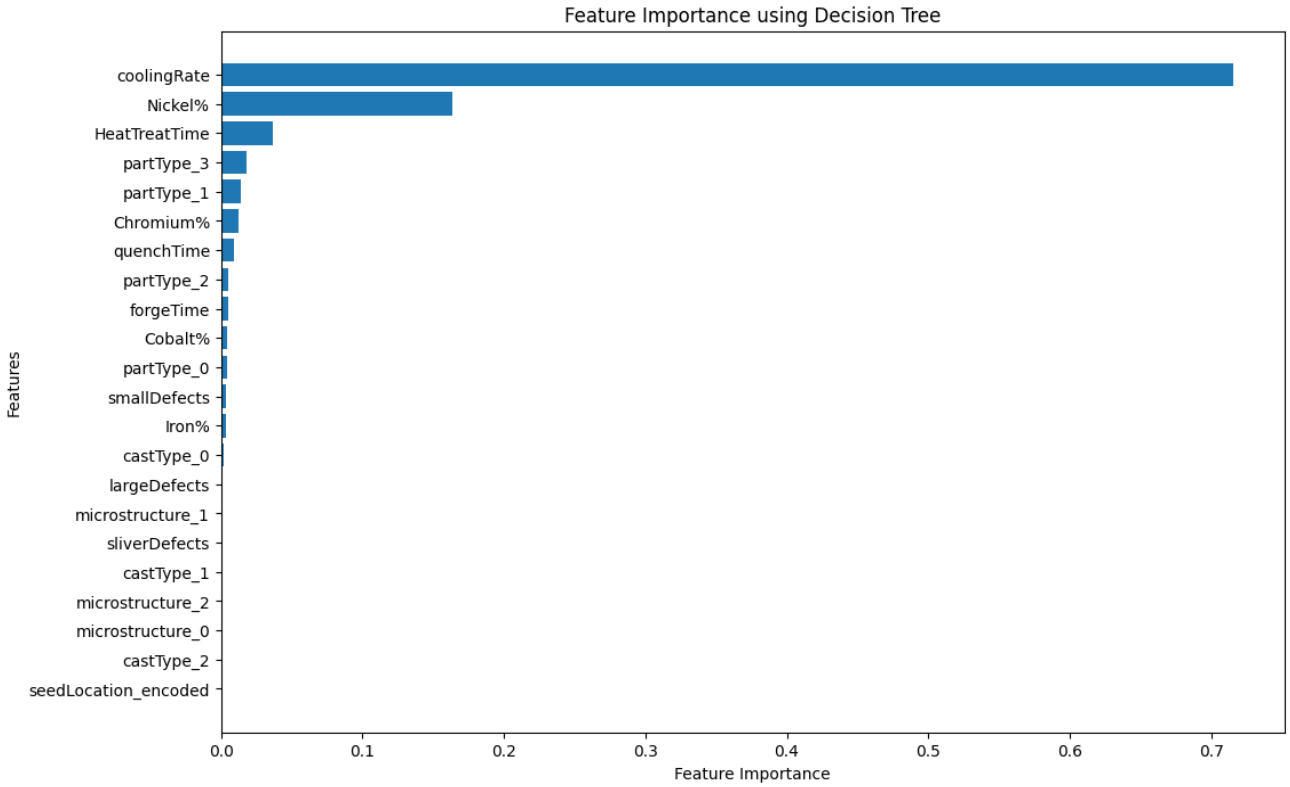

Figure 5: Feature importance based on Decision Tree

Two regression models, Gradient Boosting and MLP Neural Networks were chosen to predict the continuous numerical value of target due to their ability to capture complex, non-linear relationships in the data, crucial for predicting continuous outcomes like lifespan, as MLP Classifier and SVM with RBF Kernel were used to classify defective and non-defective metal parts. First two regression models were fitted with the top 5 important features from the decision tree and despite tuning hyperparameters, the inconsistent results of test R^2, 0.16 (Gradient Boosting) and 0.84 (NN), suggested that the decision tree-selected features were not universally optimal for different model architectures. Therefore, a combination of features from the aforementioned tests were used in model building, including “coolingRate”, “Nickel%”, “HeatTreatTime”, “partType_1”, and “Chromium%”, which yielded the best results across models (mostly above 0.90).

3. Regression Implementation

3.1 Methodology

The Gradient Boosting Regressor was selected due to its strong performance in capturing complex, non-linear relationships between features through an ensemble of trees, which reduces bias and variance compared to a single decision tree that risks overfitting. Neural Networks were also chosen for their ability to model intricate patterns in the data beyond what tree-based methods like Gradient Boosting can achieve (Towards Data Science, 2023). A Neural Network was initially implemented

using TensorFlow. Despite various configurations, including adjustments to hidden layers, neurons, and activation functions, the model's performance remained unsatisfactory, with a maximum test R^2 score of only around 0.17. Consequently, a shift was made to using MLP Regressor from Scikit-learn, which is better suited for tabular data.

The preprocessing of the five selected features involved encoding categorical variables, then the dataset was split into training, validation, and test sets using a standard 80/20 split. followed by scaling numerical features with StandardScaler to ensure consistency in feature ranges. This step is crucial for models like Neural Networks and Gradient Boosting, which are sensitive to feature scales. With a normally distributed target variable (skewness close to 0), the data was inherently well-balanced.

Gradient Boosting:

An initial fit with default Gradient hyperparameters resulted in a validation RMSE of 79.07 and R^2 of 0.945, despite indicating initial good performance, RandomizedSearchCV with 5-fold cross-validation, 50 iterations and the exact random state used for splitting data, was applied to refine the model; since experimenting and trying out every combination of parameters is an exhausting process. All the tunned hyperparameters and justifications are mentioned in table 1.

Hyperparameter	Description
N_estimators	Controls the number of boosting stages; a higher value may increase accuracy but at the cost of longer training times. The chosen range balances accuracy and efficiency.
Learning_rate	Determines the contribution of each tree to the final model; lower rates prevent overfitting but require more trees
Max_depth	Limits the depth of each tree
Subsample	Determines the fraction of samples used for fitting each tree, improving model robustness by introducing randomness.
Min_samples_split	controls the minimum number of samples required to split an internal node
Min_samples_leaf	Sets the minimum number of samples required to be in a leaf node

Table 1: Gradient Boosting Hyperparameter tuning

Multi-layer Perceptron regressor:

Similar to the previous model, first a baseline model of MLP was implemented showing negative R^2 scores of -5.20 and high error metrics. The model also triggered a Convergence Warning; likely due to having insufficient capacity, preventing it from fully learning the data patterns**.** These results indicate that the default settings were inadequate for the complexity of the dataset, hence

hyperparameters were adjusted to increase accuracy and reduce overfitting. Description of the adjusted parameters have been listed in table2.

Hyperparameter	Description
Hidden layer size	Defines the architecture of hidden layers, the number of layers and neurons in each layer. Various configurations were evaluated from small, simple to train to more complex architectures
activation	Determines how the input is transformed into output; "relu" Outputs zero for negative inputs and the input value for positive ones, and "Tanh" Outputs values between -1 and 1for prior non-linearity capturing (prone to vanishing the gradient issue).
Solver	Adjusts the weights during training; “Adam” performs well on a wide range of data, while “LBFGS” can be faster for smaller datasets with fewer updates needed
alpha	Penalizes large weights to control overfitting; higher values of alpha increase regularization, which reduces overfitting but can also decrease model flexibility
Learning_rate	Determines the pace of model’s adjustment to weight during training; “constant” keeps the rates fixed through training, “adaptive” reduces the rate when the model stops enhancing
Learning_rate_init	The initial learning rate for weights; tested at different levels to find a balance between speed and stability.
Max-iter	The maximum number of iterations the solver can take to optimize the model weights; If the model does not converge within this limit, training stops, and a warning is issued

Table 2: MLP Hyperparameter Tuning

3.2 Evaluation

GBR:

To optimize the Gradient Boosting Regressor, two sets of hyperparameter ranges were rigorously tested. The goal was to explore how adjustments to key parameters would impact model performance. The selected range values and changes are explained in tables 3 and 4.

Set 1:

Parameter	The tested range and reasoning	CV Best Value
N_estimators	50 to 300 Focused on computational efficiency proportional to learning_rate	107
Learning_rate	0.01 to 0.2 Set with a smaller learning rate as it generally leads to more robust models, albeit at the cost of requiring higher N_estimators	0.0974
Max_depth	2 to 10 Used to focus on shallow trees to improve generalization	3
Subsample	0.6 to 1.0 Used a range to balance stability and performance	0.867
Min_samples_split	2 to 20 Set to a range of maintaining a generalization and having sufficient granularity	11
Min_samples_leaf	1 to 10 A range covered to prevent splits	8

Table 3: Set 1 Tuning

Set 2:

Parameter	The tested range	CV Best Value
N_estimators	100 to 500 expanded the range to evaluate whether additional trees could further improve performance	143
Learning_rate	0.005 to 0.1	0.0789

	Set to a tighter range comparing to the previous model to enhance regularization and prevent overfitting	
Max_depth	3 to 15 Expanded the range to find out whether deeper trees enhance the model	3
Subsample	0.7 to 1.0 Increased to find further enhancement	0.736
Min_samples_split	5 to 30 Expanded the range to allow exploration of higher values	23
Min_samples_leaf	2 to 15 Expanded range to emphasize generalization by limiting granularity	12

Table 4: Set 2 tuning

Despite the adjustments, both models achieved identical performance metrics, indicating that the model had likely reached its optimal performance limit with the dataset. The adjustments in learning rate and tree depth optimized the model to learn efficiently without overfitting. However, increasing the number of estimators and adjusting the subsample fraction did not lead to noticeable improvements, implying that the model was already robust. To assess model performance, the following metrics were utilized; R^2 indicates how well the model explains the data variance, while MAE provides the average prediction error. According to table 5, MSE and RMSE emphasize larger errors, with RMSE being particularly useful for interpreting errors in the original units.

Metric	Set 1	Set 2
Test R^2	0.9508	0.9508
Test MAE	62.72	62.72
Test MSE	5903.58	5903.58
Test RMSE	76.83	76.83

Table 5: GB Performance on Test Set

The second set (with slightly more conservative settings) is recommended for deployment, due to its enhanced regularization, ensuring better performance on unseen data. This model's consistency across different hyperparameter settings indicate that it generalizes well, making it a reliable choice for predicting metal part longevity. Referring to fig 6, most points align closely with the diagonal line, reflecting accurate predictions and a high degree of consistency. However, slight deviations in extreme ranges suggest opportunities for fine-tuning to improve performance further, particularly for edge cases.

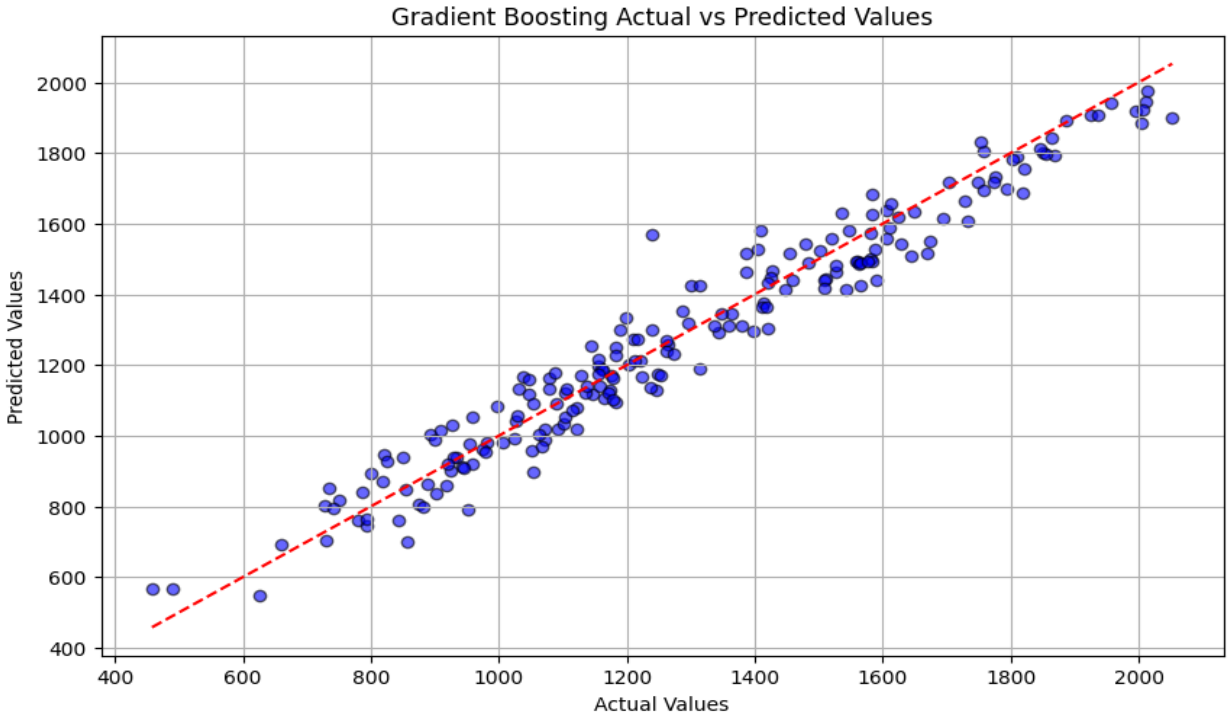

Figure 6

MLP:

The same cross validation approach was used on the MLP Regressor. Using the best values found, test data was used for final evaluation. It performed best with a single hidden layer of 100 neurons, leveraging the "Tanh" activation function to capture non-linear patterns effectively. An adaptive learning rate starting at 0.037872 enabled efficient convergence without overshooting, balancing complexity and performance (table 6).

This model achieves strong predictive accuracy with an R^2 score of **0.917** and according to the scatter plot, most points are aligned with the red diagonal line. However, there is a tendency to overpredict at lower values and underpredict at higher values (table 7).

Hyperparameter	Range	CV Best Value
Hidden layer size	[(50,), (100,), (50, 30), (100, 50), (100, 50, 30)] This range allows testing both small configurations to prevent overfitting and deeper networks for capturing complex patterns	(100,)
activation	ReLU prevents the vanishing gradient issue and works well with deeper networks. Tanh is useful for centring data, which aids Normally distributed models	“tanh”
Solver	Compares Adam's dynamic learning with LBFGS's quick convergence for smaller datasets	“adam”
alpha	0.0001 to 0.01 Testing a range of values: lower ones fitting the training data well and higher values regularizing the model	0.0032720
Learning_rate	Constant ensures stability throughout training and Adaptive decreases when progress stalls	“adaptive”
Learning_rate_init	0.001 to 0.1 a range testing smaller values helping gradual learning and higher ones speeding learning	0.037872

Table 6: MLP Best Hyperparameters and Reasoning

Metric	MLP Regressor
Test R^2	0.917
Test MAE	80.69
Test MSE	10235.23
Test RMSE	101.17

Table 7: MLP Performance on Test Set

As plotted in fig 7, the model has successfully captured the overall relationship. however, there are deviations at the lower and higher ends of the lifespan range, where the model struggles. Overprediction is seen for lower lifespan values, while underprediction occurs at higher lifespan values.

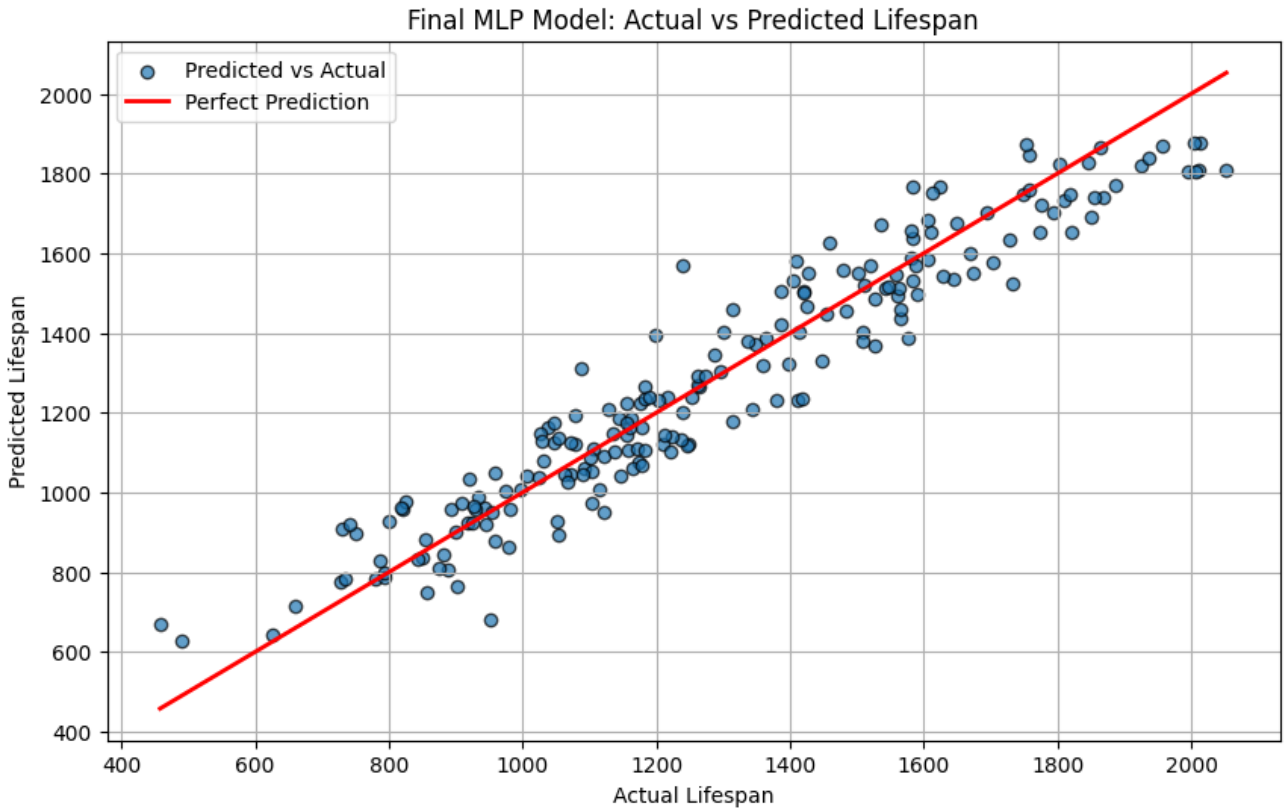

Figure 7

Final Comparison:

The GB model makes fewer larger errors and explains more variance especially in the mid-range of the data in the target along with higher accuracy, as the MLP struggles with the edges, leading to higher residuals; making GBR the recommended model due to its robustness and well generalization. The MLP Regressor, while effective, is more sensitive to outliers. However, it remains valuable for datasets with pronounced non-linear relationships or future data expansions (table 8).

Metric	Gradient Boosting	MLP Regressor
Test R^2	0.9508	0.917
Test MAE	62.72	80.69
Test MSE	5903.58	10235.23
Test RMSE	76.83	101.17

Table 8: Final Comparison

3.3 Critical Review

A fair comparison was assured through data preprocessing which also minimized data leakage. Minimal Hyperparameter tuning achieved high accuracy in the GBR and also captured non-linear relationships effectively, particularly for mid-range values, showcasing its flexibility for complex patterns. However, both models struggled with extreme values, as evidenced by higher residuals and RMSE at the dataset's tails. They could benefit from outlier removal or transformation. RandomizedSearchCV, while efficient, limits hyperparameter exploration; advanced methods like Bayesian Optimization could further optimize model configurations.

4. Classification Implementation

4.1 Feature Crafting

To better categorize metal parts' lifespans, instead of using a simplistic binary threshold at 1500 hours, a more sophisticated approach using K-means clustering (unsupervised ML technique) was applied; the aim was to uncover underlying groupings. The best K number was searched through the Elbow and Davies-Bouldin methods after encoding and scaling the features. The Elbow Method plots the inertia for different k values; optimal k is where the curve flattens. However, it just suggests the potential values of k = 5, 6, 7. Therefore, Davies Bouldin was used to calculate the ratio of within-cluster dispersion to the separation between clusters; Lower values indicating better quality which is k = 5, figures 8 and 9.

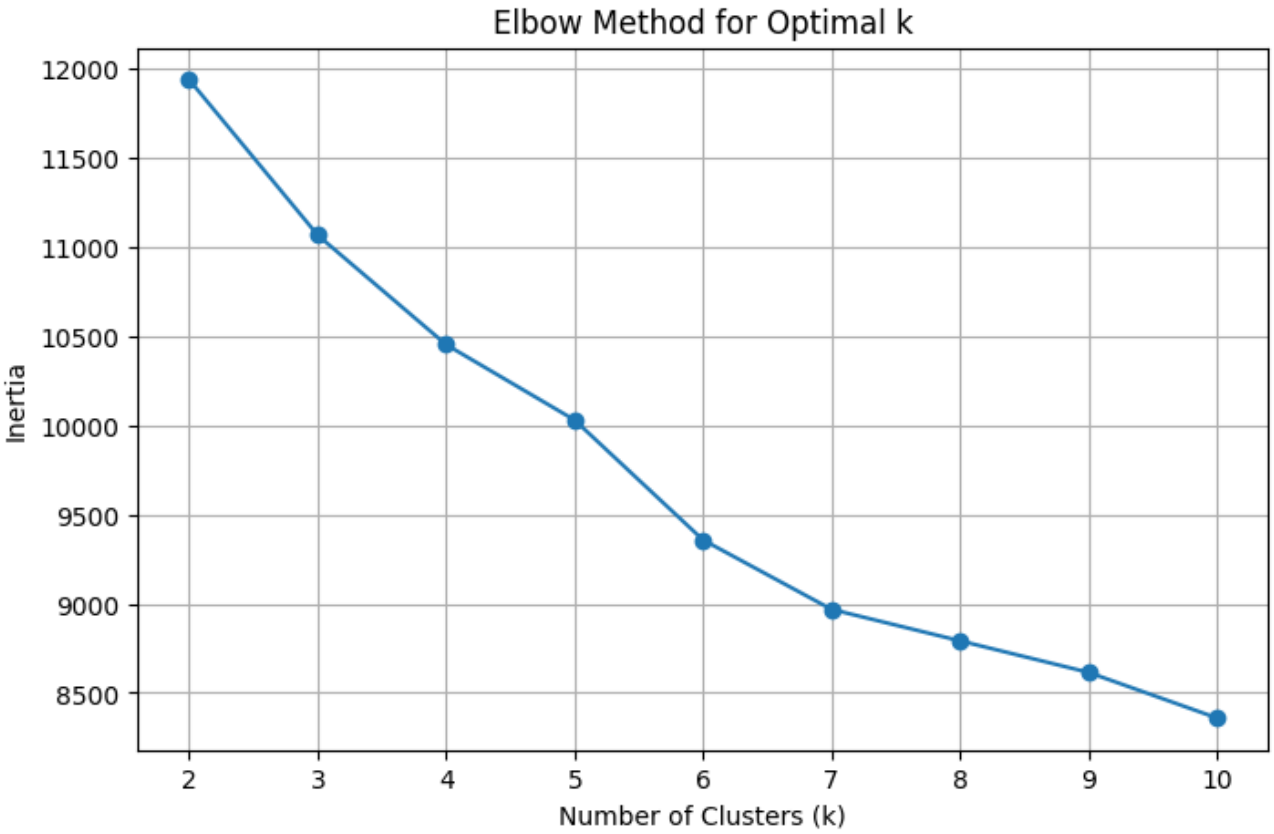

Figure 8

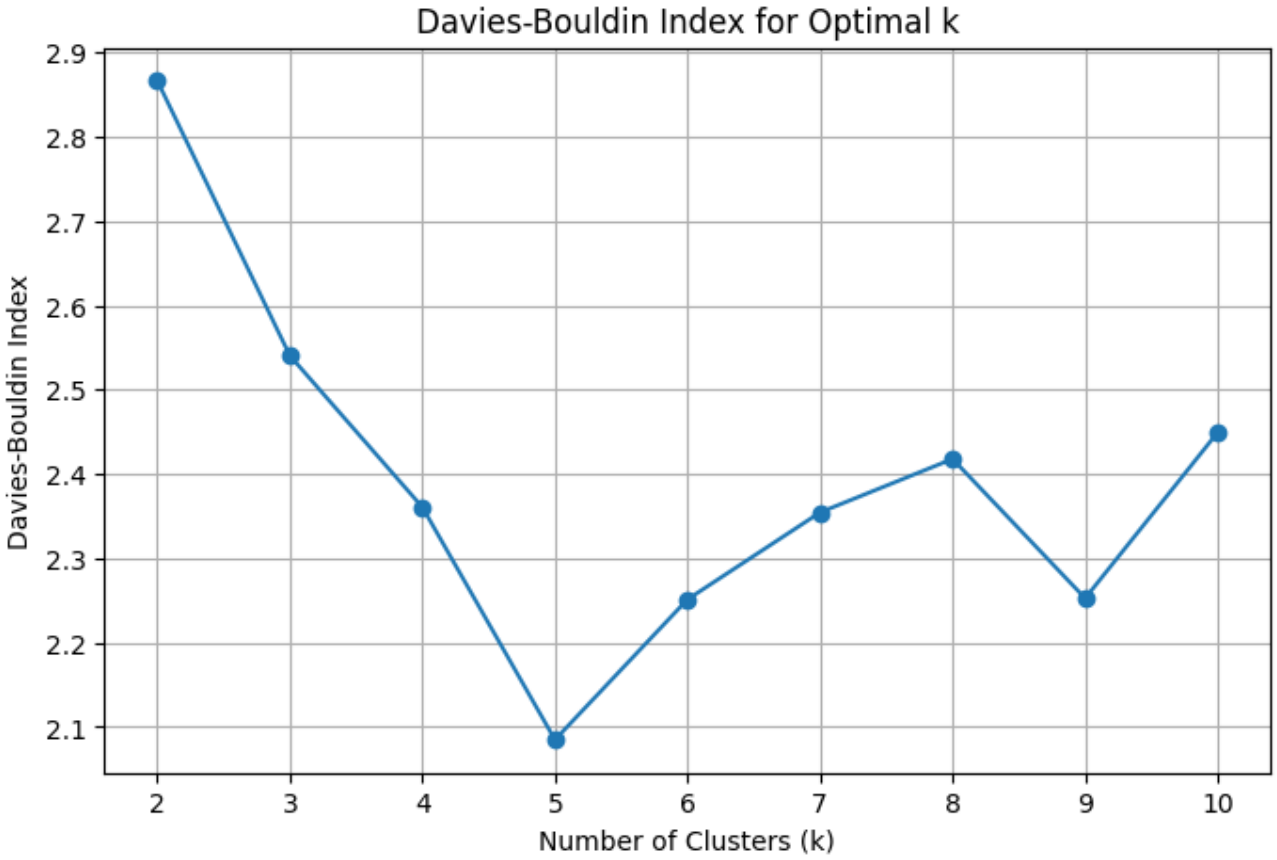

Figure 9

Illustrated in fig10, PCA reduced the dataset to two components, capturing most of the variance for easier visualization; the clusters are color-coded, showing a generally well-separated structure with some overlap, particularly between orange/green and blue/purple clusters. Clusters like purple and red appear more compact and distinct, while others, such as green and orange, are more spread

out, indicating higher variance within those groups.

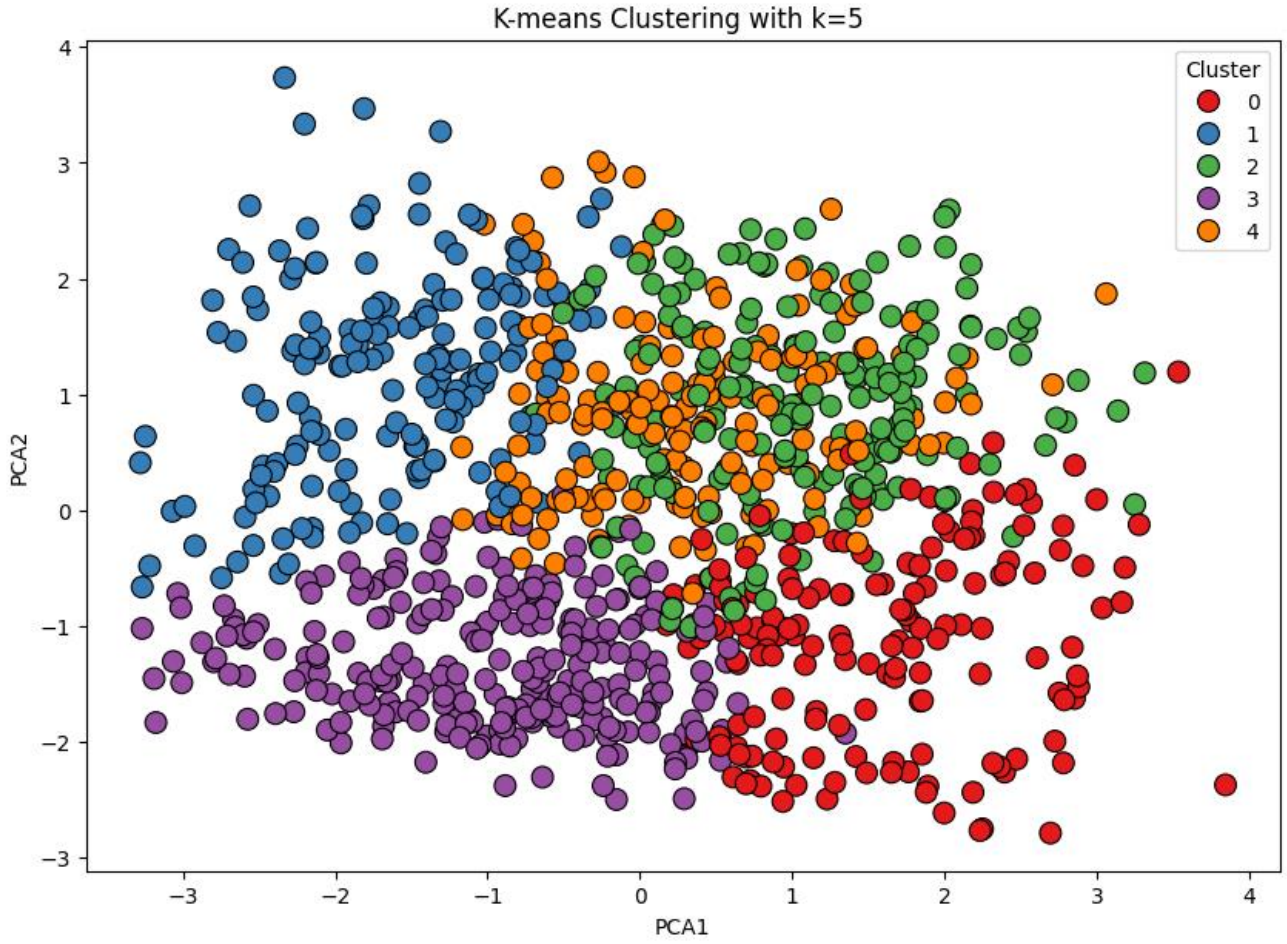

Figure 10

Using a binary threshold at 1500 hours, fig 11, splits the dataset into two groups: "Defective" (lifespan <1500 hours) and "non-defective" (lifespan ≥1500 hours). However, this approach is simplistic and fails to capture the variability in lifespans. Unlike thresholding, clustering (fig 12) avoids oversimplification and ensures fair representation of all groups, with cluster sizes ranging from 17.9% to 25.1%.

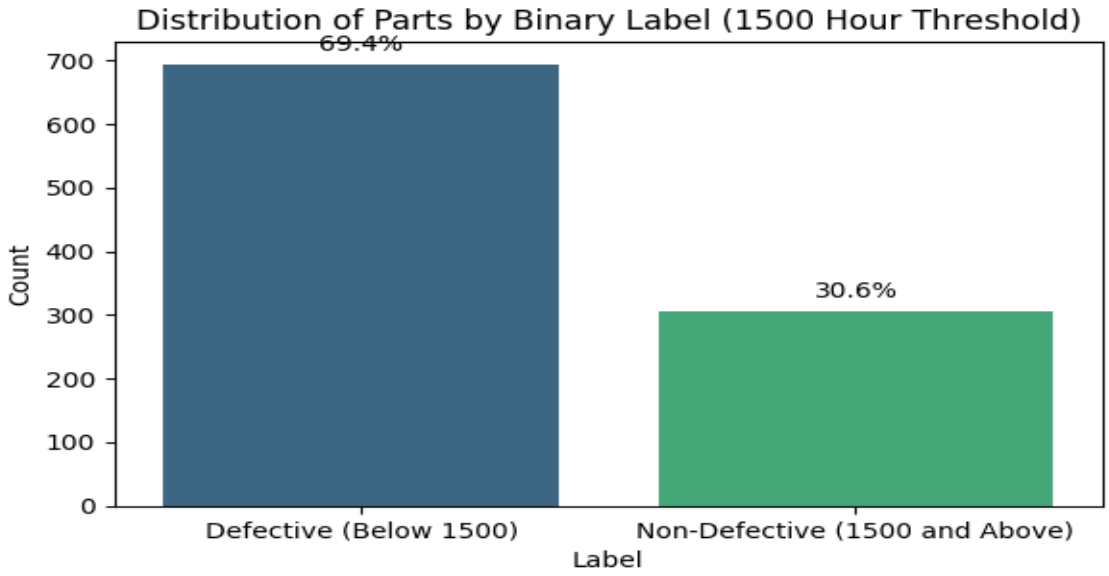

Figure 11

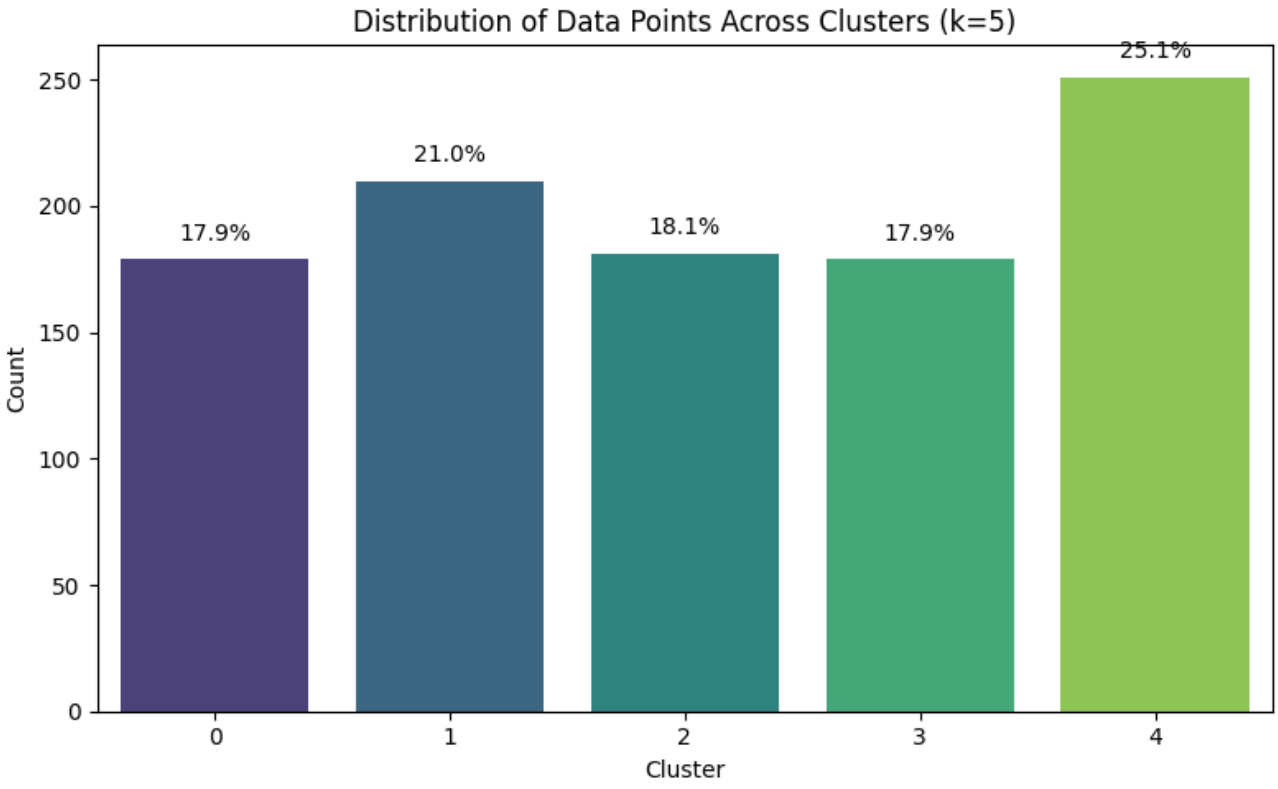

Figure 12

4.2 Methodology

Given the multi- dimensional, non-linear and complex nature of the data MLP Classifier and SVM with RBf Kernel were chosen (Seinfeld, Blom and Karahagopian, 1997); MLP relies on neural

network-based learning, while SVM emphasizes decision boundaries. This diversity ensures a thorough exploration of classification performance. To understand what each cluster represents, the mean values of features within each cluster were analyzed in table 9:

Cluster Labels	Description
0	Short lifespan, possible defects
1	Intermediate lifespan, balanced quality
2	Long lifespan, durable parts
3	Short lifespan, poor quality
4	Moderate lifespan, average quality

Table 9: Labelling

To preprocess the data “1500_labels” and “Lifespan” were deleted from the copy of dataset as “Cluster_Labels” is the new target. The same procedure of encoding, splitting and scaling was done on the features. According to figure 12, the clustering did a fair job of dividing data except for a slight imbalance in the cluster’s size in k = 4. Initially a baseline of both models was fitted; with 0.65 MLP and 0.92 SVM validation accuracy. Same cross validation process using RandomizedSearchCV with 50 iterations for MLP and 20 for SVM values, 5 folds and the same random seed was applied to tune the models. The table below provides the range and description of tunned MLP hyperparameters (refer to table 6 for the repeated parameters), followed by SVM’s parameter chart (tables 10 and 11).

MLP Hyperparameter	Range and Description
Hidden layer size	(50,), (100,), (150,), (200,) --Single layer (50, 30), (100, 50), (150, 100) --Two layers (50, 30, 10), (100, 50, 30), (200, 100, 50) --Three layers Tests a range of various complexities
activation	identity: Linear transformation for baseline comparison logistic: Suitable for probability outputs and smooth transitions tanh: Zero-centered function, better for data with negative and positive ranges relu: Handles vanishing gradients and is efficient for deep networks
Solver	adam: effective for large datasets and non-convex problems sgd: sensitive to learning rates and momentum lbfgs: Converges faster for small datasets, suitable for optimization problems
alpha	[0.0001, 0.001, 0.01, 0.1, 1]

	Covers underfitting and overfitting
Learning_rate	Constant: Fixed learning rate for stable training. Adaptive: Reduces learning rate when no improvement is observed Invscaling; ensuring improving when progress slows
Learning_rate_init	[0.0001, 0.001, 0.01, 0.1] Tests gradual vs faster learning pace
early_stopping	[True] Prevents overfitting when validation performance stalls
n_iter_no_change	[10] Stops training after 10 stagnant iterations, preventing wasted computational resources
Max_iter	[500, 1000, 2000] Sets the maximum number of training iteration; higher values allow converging fully
momentum	[0.9, 0.95, 0.99] helps speeding convergence (Works only on sgd solver); tests a range of gradual vs fast convergence

Table 10

SVM Hyperparameter	Range and Description
C	[0.001, 0.01, 0.1, 1, 10, 100, 1000] Maintains a balance between achieving low error on the training data and model complexity; The range above covers overfitting and underfitting
Gamma	[0.001, 0.01, 0.1, 0.5, 1, 5, 10, 50] Sets the effect of one single training example; the range covers local and global boundaries
Kernel	[rbf] Captures non-linear patterns

Table 11

4.3 Evaluation

MLP:

During the first set of tunning, a lbfg convergence warning was encountered within the specified max_iter=1000 although this value was included in the parameter grid. During CV, some combinations of hyperparameters might have led the model to struggle with convergence; activation functions like identity or high regularization (alpha) values might make it harder for the optimization to reach a solution within the set iterations. Therefore, a second and third tunning were applied by setting iteration to 2000 and 3000 along the best parameters initially found; resulting in the previous lbgf warning and no improvement. To address this the solver was changed to 'adam' (forth set) which dropped test accuracy significantly; revealing that it struggles with class imbalance. Set 1 remains the best model because it achieves the same results as Set 2 but with fewer computational resources (1000 iterations compared). Accuracy measures correctness, precision focuses predictions, recall identifies true positives and F1 balances precision and recall in table 12.

MLP Hyperparameter	Best CV set1	Set 2 tunning change	Set 3 tunning change	Set 4 tunning change
Hidden layer size	(200,)			
activation	Identify			
Solver	Lbfgs			adam
alpha	1			
Learning_rate	Constant			
Learning_rate_init	0.01			
early_stopping	True			
n_iter_no_change	20			
Max_iter	1000	2000	3000	
momentum	0.9			

Test Set 1 Evaluation:	Set 2 and 3	Set 4
Accuracy: 0.7000	0.7000	0.4400
Precision: 0.7105	0.7105	0.6309
Recall: 0.7000	0.7000	0.4400
F1 Score: 0.7032	0.7032	0.4031

Table 12: (empty boxes mean no change in the value)

As shown in fig 13, model classifies 49 values correctly in class 3, but there's 15 misclassifications as class 0; showing the overlap of class 0 and 3. Class 2 has acceptable performance with some misclassification of neighbouring classes 0 and 1. 12 misclassifications in class 1 as 2 and 11 as

class 3, shows difficulty separating the classes. The model achieves a 70% accuracy, with significant misclassifications. Precision (71%) indicates that most predictions are correct, while recall (70%) shows the model successfully identifies most true instances, but struggles to distinguish overlapping classes, particularly between Classes 0 and 3 (Fawcett, 2006).

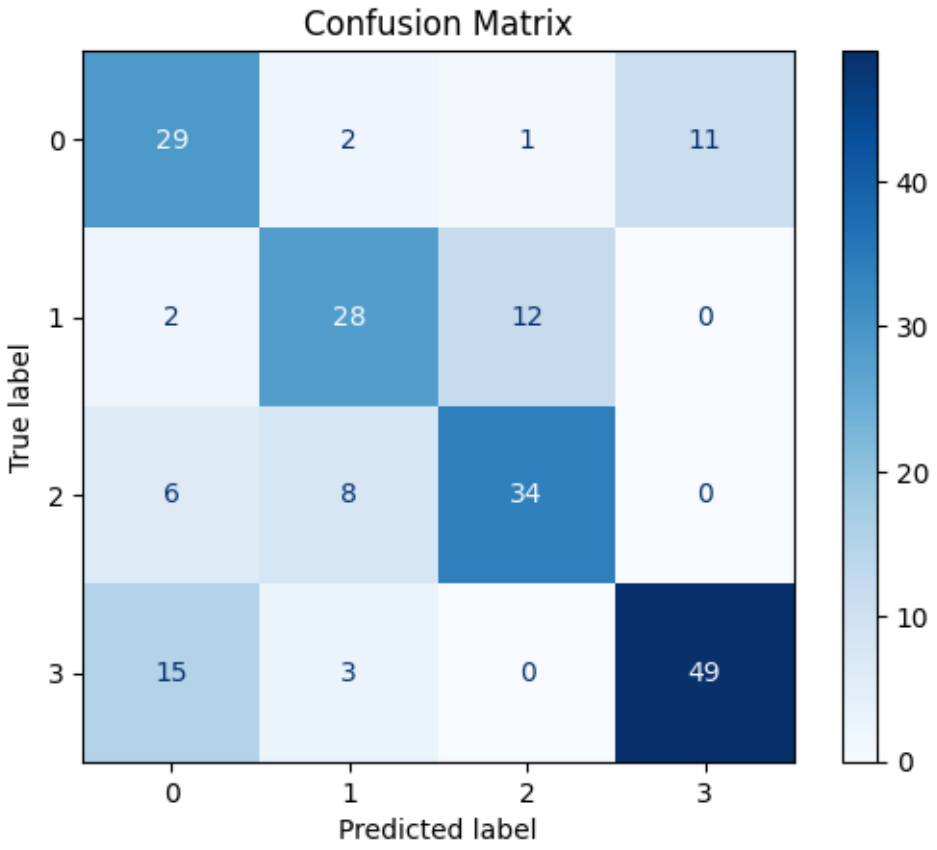

Figure 13

SVM:

Fitting the final model on test data using the values from table 13, led to an accurate model. According to table 14, these metrics indicate that the model performs well, achieving high accuracy and balanced precision, recall, and F1 score. The strong F1 score highlights the model's ability to balance precision (minimizing false positives) and recall (minimizing false negatives). The confusion matrix in fig 14, reflects excellent classification accuracy across all clusters, with minimal misclassification errors being 5 instances of Class 3 misclassified as Class 0.

SVM Hyperparameter	Best CV
C	1000
Gamma	0.01
Kernel	rbf

Table 13

Test Accuracy	0.9450
Test Precision	0.9500
Test Recall	0.9450
Test F1 Score	0.9455

Table 14

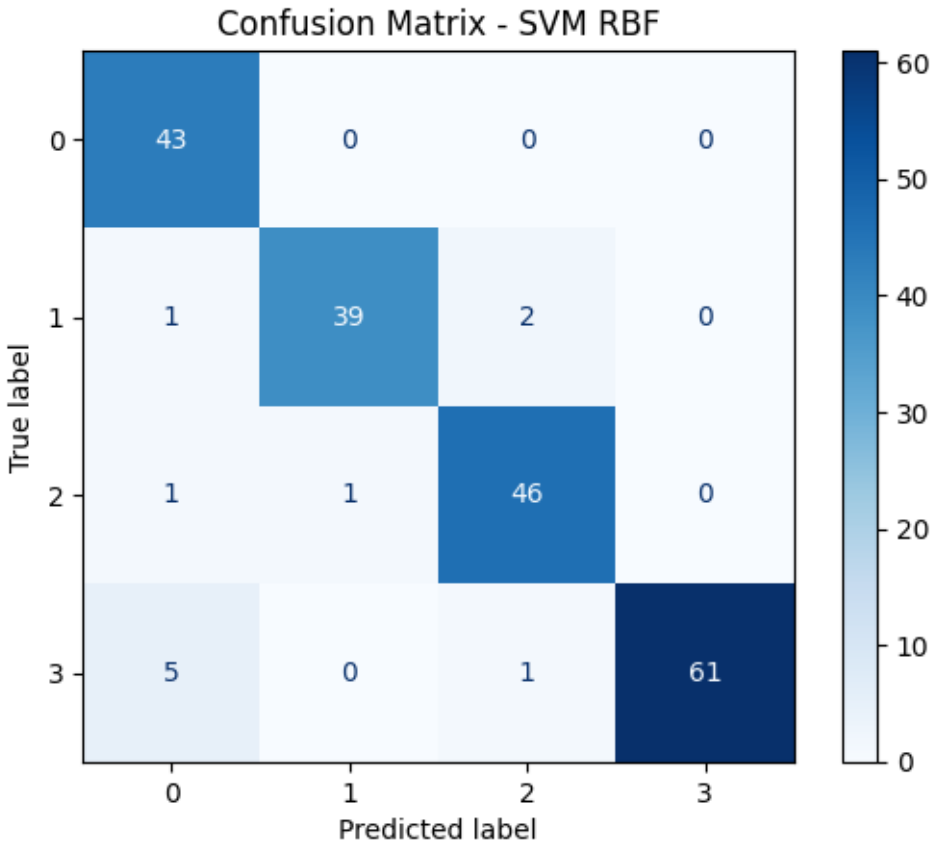

Figure 14

Final comparison:

It's clear from table 15 that the SVM RBF model outperforms MLP, effectively capturing complex, non-linear relationships while remaining robust against overfitting.

Final Results	MLP	SVM
Accuracy	0.7000	0.9450
Precision	0.7105	0.9500
Recall	0.7000	0.9450
F1 Score	0.7032	0.9455

Table 15

4.4 Critical Review

The use of optimal values for C (1000) and Gamma (0.01) enables the model to strike a balance between margin maximization and decision boundary complexity. Exploring SVM kernels, polynomial degree 2 achieved 0.90 accuracy, proving kernel is the better approach.

Overlap between specific MLP classes suggests that the model struggles to capture subtle differences in the feature space. Using weighted loss function, advanced feature engineering, better clustering, and model tuning could leave room for improvement.

5. Conclusions

The findings from the experiments align with the initial observations during data exploration, which indicated the need for models that can handle non-linear relationships. The SVM with RBF kernel emerged as the best-performing model for classification, achieving 94.5% accuracy with balanced precision, recall, and F1 scores, effectively classifying parts into usable and defective categories with minimal errors.

On the other hand, the Gradient Boosting Regressor achieved an R^2 of 0.9508, demonstrating its strength in predicting part lifespan with high accuracy. Both models are robust and well-suited for their respective tasks. However, considering the simplicity and reliability of classifying usability, the SVM model is recommended for deployment.

6. References

1. **Fawcett, T., 2006.** An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8), pp.861-874. Available at: https://doi.org/10.1016/j.patrec.2005.10.010 [Accessed 11 Nov. 2024].
2. **Haykin, S., 1999.** *Neural Networks: A Comprehensive Foundation*. 2nd ed. Prentice Hall.
3. **Seinfeld, S., Blom, K. and Karahagopian, V., 1997.** Assessing the effectiveness of urban air quality management using machine learning. *Atmospheric Environment*, 32(20), pp.3603-3611. Available at:

https://www.sciencedirect.com/science/article/abs/pii/S1352231097004470 [Accessed 10 Nov. 2024].

4. **4.Towards Data Science, 2023.** Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis. *Towards Data Science*. Available at: https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141 [Accessed 2 Nov. 2024].